Advanced Hedging Strategies in Volatile Markets: A Quantitative Comparison of Delta and Vega Hedging Performance Under Stochastic Volatility Models

Table of Contents

Introduction

In conventional options pricing theory, the Black-Scholes model makes the assumption that volatility is constant, which is not true in real markets where volatility varies dynamically. The performance of conventional delta hedging and more advanced vega hedging techniques is compared in this study to examine the critical impact of stochastic volatility on options hedging strategies.

We use these hedging methods and examine them over thousands of simulated pathways using Monte Carlo simulations with three distinct volatility models (Custom Switching, Heston, and GARCH). Our study bridges the gap between academic finance and real-world trading applications by offering traders realistic insights that go beyond theoretical frameworks. These insights include interactive visualizations and professional-grade analytics tools.

In addition to presenting quantitative data about the circumstances under which vega hedging becomes important in volatile market conditions, the study illustrates the impact of volatility regimes, mean-reversion features, and volatility

clustering on hedging performance. Assume that the implied volatility (IV) of a one-year call option is 20%. Even if there is no change in the stock price, your short option will cost more if IV spikes up to 30%. Vega hedging guards against this by shorting a second option whose vega balances your initial position (for example, a different strike/expiry). Because it avoids blowups during volatility spikes, vega hedging is therefore similar to purchasing "volatility insurance" despite being more expensive to maintain.

In [26]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from mpl_toolkits.mplot3d import Axes3D
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import matplotlib.animation as animation
from IPython.display import HTML
from scipy.optimize import minimize
sns.set_style('darkgrid')
from scipy.linalg import cholesky
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from tabulate import tabulate
```

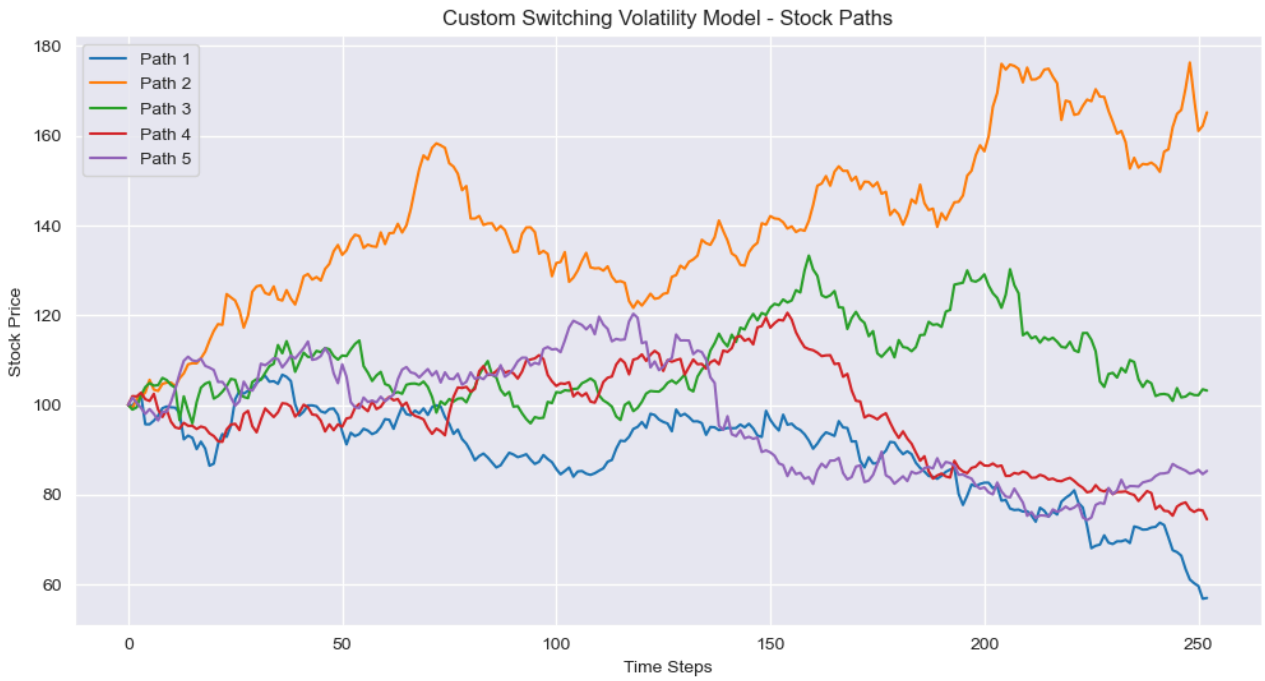
1. Volatility Modeling Framework

We will implement three different volatility models to generate realistic stock paths:

1.1 Custom Switching Volatility Model

In [18]:
```
def stock_path_custom_sigma(S0, t, r, mu, n_paths, n_steps, vol_clusters=None):
    if vol_clusters is None:
        vol_clusters = {
            'vols': [0.2, 0.3, 0.45],
            'probs': [0.5, 0.3, 0.2],
            'persistence': 0.7
        }

    noise = np.random.normal(0, 1, size=(n_paths, n_steps))
    dt = t/n_steps

    sigma = np.zeros((n_paths, n_steps))
    for i in range(n_paths):
        current_regime = np.random.choice(
            len(vol_clusters['vols']),
            p=vol_clusters['probs']
        )
        sigma[i, 0] = vol_clusters['vols'][current_regime]

        for j in range(1, n_steps):
            if np.random.rand() < vol_clusters['persistence']:
                sigma[i, j] = sigma[i, j-1]
            else:
                sigma[i, j] = np.random.choice(
                    vol_clusters['vols'],
                    p=vol_clusters['probs']
                )

    increments = (mu + r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)*noise
    log_returns = np.cumsum(increments, axis=1)
    paths = S0*np.exp(log_returns)
    paths = np.insert(paths, 0, S0, axis=1)

    return paths, sigma

paths, sigma = stock_path_custom_sigma(S0=100, t=1, r=0.03, mu=0.08, n_paths=5, n_steps=252)
```

```
plt.figure(figsize=(12, 6))
for i in range(5):
    plt.plot(paths[i], label=f'Path {i+1}')
plt.title("Custom Switching Volatility Model - Stock Paths")
plt.xlabel("Time Steps")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```



This model shows abrupt volatility regime changes creating discontinuous jumps in stock paths.

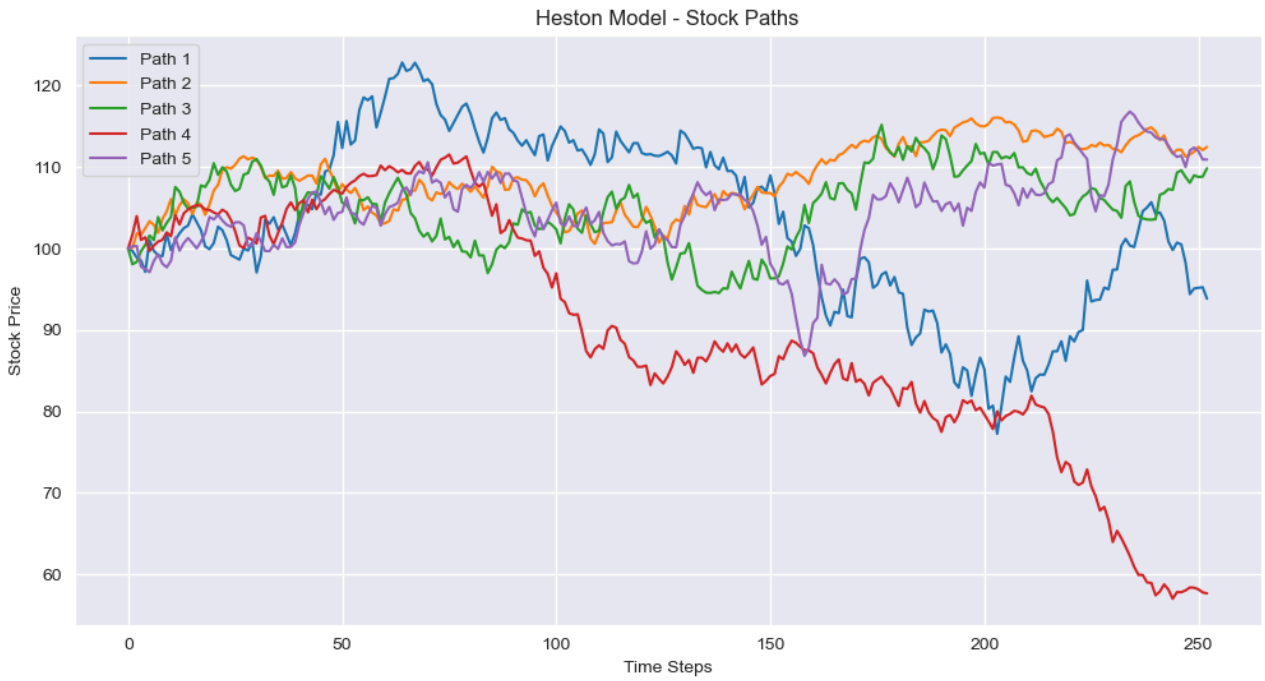
Regime switching probability: 30% chance to change volatility states.

1.2 Heston Model

In [20]:

```
def heston_model(S0, t, r, kappa, theta, xi, rho, n_paths, n_steps, v0=None):
    if v0 is None:
        v0 = theta

    dt = t/n_steps
    paths = np.zeros((n_paths, n_steps+1))
    vol_paths = np.zeros((n_paths, n_steps+1))

    paths[:, 0] = S0
    vol_paths[:, 0] = v0

    for i in range(1, n_steps+1):
        Z1 = np.random.normal(0, 1, n_paths)
        Z2 = rho*Z1 + np.sqrt(1-rho**2)*np.random.normal(0, 1, n_paths)

        vol_paths[:, i] = vol_paths[:, i-1] + kappa*(theta - np.maximum(vol_paths[:, i-1], 0))*dt + \
                          xi*np.sqrt(np.maximum(vol_paths[:, i-1], 0)*dt)*Z1

        vol_paths[:, i] = np.maximum(vol_paths[:, i], 0)

        paths[:, i] = paths[:, i-1]*np.exp(
            (r - 0.5*vol_paths[:, i-1])*dt +
            np.sqrt(vol_paths[:, i-1]*dt)*Z2
        )

    return paths, np.sqrt(vol_paths)

paths, sigma = heston_model(S0=100, t=1, r=0.03, kappa=2, theta=0.2**2, xi=0.3, rho=-0.7, n_paths=5, n_s
```

```
plt.figure(figsize=(12, 6))
for i in range(5):
    plt.plot(paths[i], label=f'Path {i+1}')
plt.title("Heston Model - Stock Paths")
plt.xlabel("Time Steps")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```



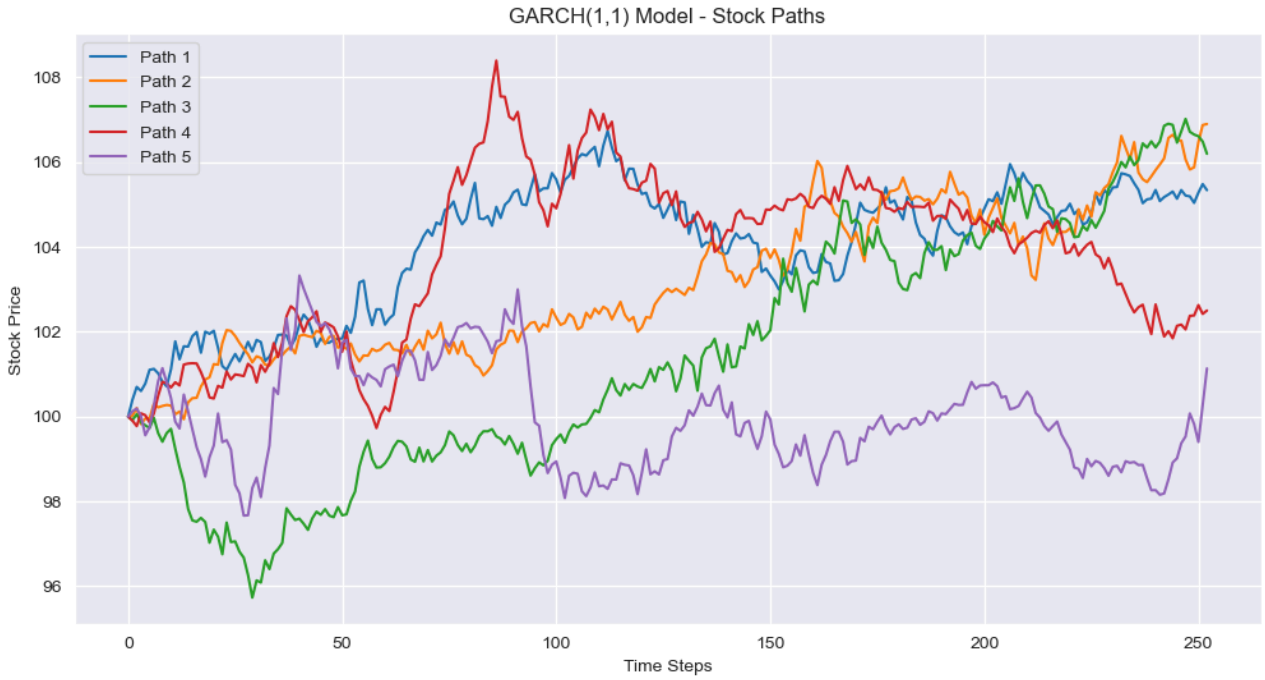
This model demonstrates smoother mean-reverting volatility with negative correlation to price (-0.7).

Heston parameters: $\kappa = 2$ (fast mean reversion), $\theta = 0.04$ (long-term variance).

1.3 GARCH(1,1) Model

In [21]:

```
def garch_model(S0, t, r, n_paths, n_steps, omega=0.0001, alpha=0.1, beta=0.85):
    dt = t/n_steps
    paths = np.zeros((n_paths, n_steps+1))
    vol_paths = np.zeros((n_paths, n_steps+1))

    paths[:, 0] = S0
    vol_paths[:, 0] = np.sqrt(omega/(1 - alpha - beta))

    for i in range(1, n_steps+1):
        returns = np.random.normal(0, 1, n_paths) * vol_paths[:, i-1] * np.sqrt(dt)
        paths[:, i] = paths[:, i-1] * np.exp((r - 0.5*vol_paths[:, i-1]**2)*dt + returns)

        squared_return = (np.log(paths[:, i]/paths[:, i-1]) - (r - 0.5*vol_paths[:, i-1]**2)*dt)**2 / dt
        vol_paths[:, i] = np.sqrt(omega + alpha*squared_return + beta*vol_paths[:, i-1]**2)

    return paths, vol_paths

paths, sigma = garch_model(S0=100, t=1, r=0.03, n_paths=5, n_steps=252)
plt.figure(figsize=(12, 6))
for i in range(5):
    plt.plot(paths[i], label=f'Path {i+1}')
plt.title("GARCH(1,1) Model - Stock Paths")
plt.xlabel("Time Steps")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```

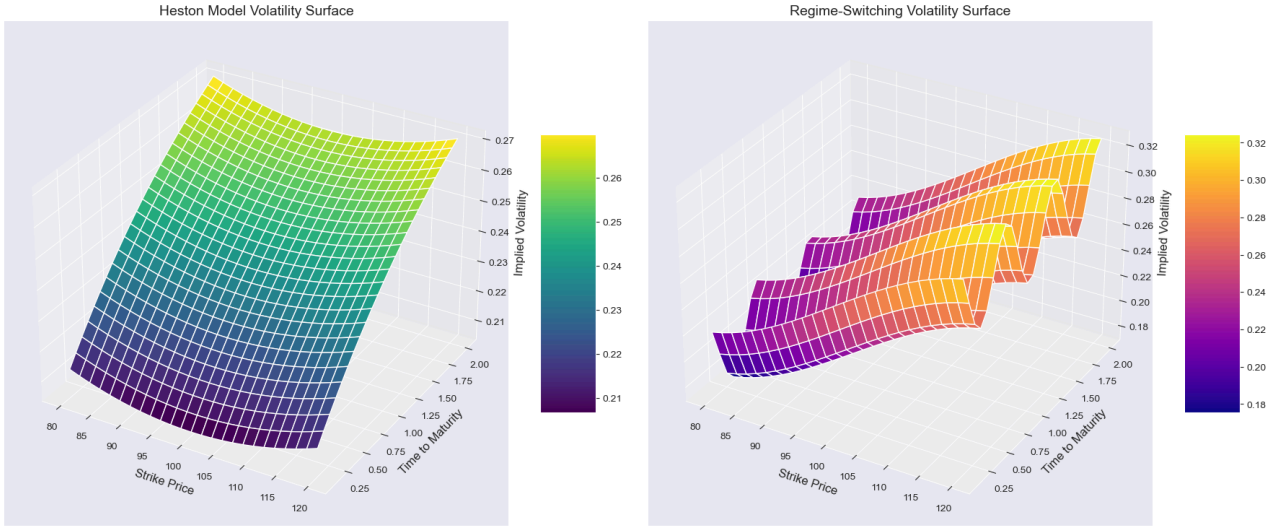


This model exhibits volatility clustering with persistent high-volatility periods.

GARCH parameters: $\alpha = 0.1$ (news impact), $\beta = 0.85$ (persistence).

1.4 Volatility Surface Visualization

In [5]:
```
def plot_volatility_surfaces():
    S0=100
    strikes = np.linspace(80, 120, 25)
    maturities = np.linspace(0.1, 2, 25)
    X, Y = np.meshgrid(strikes, maturities)

    Z_heston = 0.2 + 0.1*(1 - np.exp(-0.5*Y)) + 0.2*(X/S0 - 1)**2

    Z_switch = 0.25 + 0.05*np.sin(2*np.pi*(X/S0)) + 0.03*np.cos(3*np.pi*Y)

    fig = plt.figure(figsize=(18, 8))

    ax1 = fig.add_subplot(121, projection='3d')
    surf1 = ax1.plot_surface(X, Y, Z_heston, cmap='viridis')
    ax1.set_title('Heston Model Volatility Surface', fontsize=14)
    ax1.set_xlabel('Strike Price', fontsize=12)
    ax1.set_ylabel('Time to Maturity', fontsize=12)
    ax1.set_zlabel('Implied Volatility', fontsize=12)
    fig.colorbar(surf1, ax=ax1, shrink=0.5, aspect=5)

    ax2 = fig.add_subplot(122, projection='3d')
    surf2 = ax2.plot_surface(X, Y, Z_switch, cmap='plasma')
    ax2.set_title('Regime-Switching Volatility Surface', fontsize=14)
    ax2.set_xlabel('Strike Price', fontsize=12)
    ax2.set_ylabel('Time to Maturity', fontsize=12)
    ax2.set_zlabel('Implied Volatility', fontsize=12)
    fig.colorbar(surf2, ax=ax2, shrink=0.5, aspect=5)

    plt.tight_layout()
    plt.show()

plot_volatility_surfaces()
```



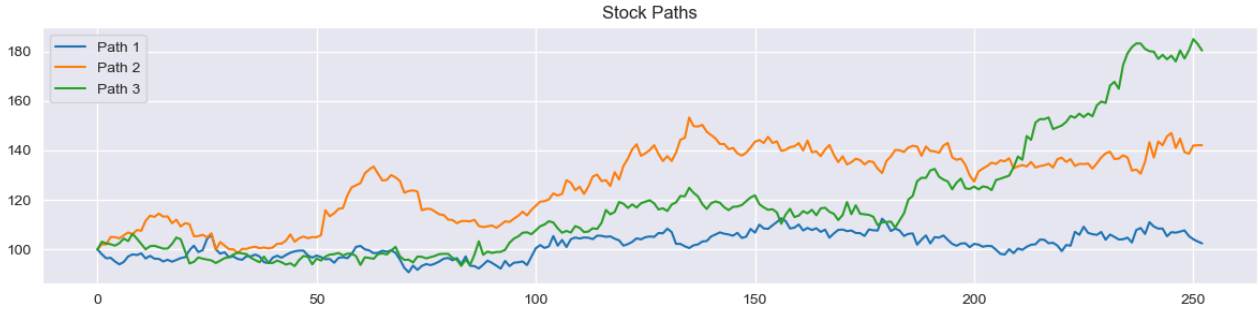
2. Delta Hedging

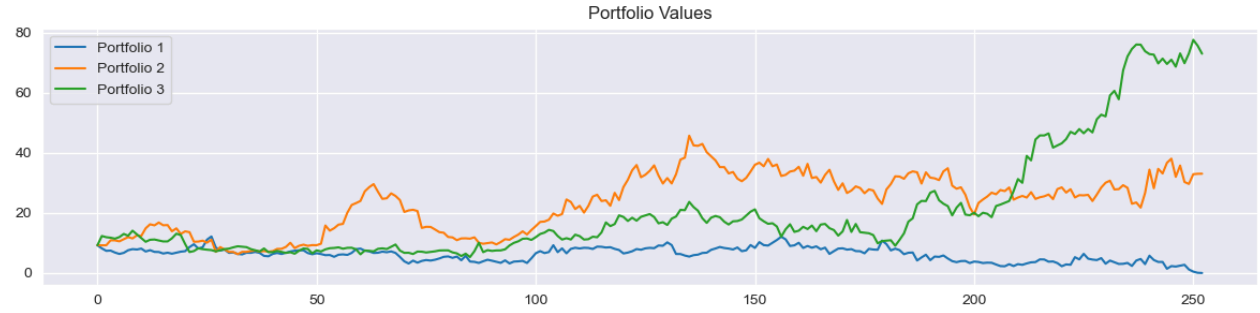
In [27]:
```
def black_scholes(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)

    if option_type == 'call':
        price = S*norm.cdf(d1) - K*np.exp(-r*T)*norm.cdf(d2)
        delta = norm.cdf(d1)
    else:
        price = K*np.exp(-r*T)*norm.cdf(-d2) - S*norm.cdf(-d1)
        delta = -norm.cdf(-d1)

    return price, delta

def delta_hedge_simulation(S0, K, T, r, mu, sigma, n_steps, n_paths, model='custom'):
    if model == 'custom':
        paths, actual_sigma = stock_path_custom_sigma(S0, T, r, mu, n_paths, n_steps)
    elif model == 'heston':
        paths, actual_sigma = heston_model(S0, T, r, kappa=2, theta=0.2**2, xi=0.3, rho=-0.7,
                                           n_paths=n_paths, n_steps=n_steps)
    else:
        paths, actual_sigma = garch_model(S0, T, r, n_paths=n_paths, n_steps=n_steps)

    dt = T/n_steps
    time_points = np.linspace(0, T, n_steps+1)

    portfolio_values = np.zeros((n_paths, n_steps+1))
    hedge_errors = np.zeros((n_paths, n_steps+1))

    initial_price, initial_delta = black_scholes(S0, K, T, r, sigma[0,0] if model != 'custom' else sigma

    for i in range(n_paths):
        cash = initial_price - initial_delta * S0
        portfolio_values[i, 0] = cash + initial_delta * paths[i, 0]

        for j in range(1, n_steps+1):
            time_remaining = T - time_points[j]

            current_vol = actual_sigma[i, j-1] if model != 'custom' else actual_sigma[i, j-1]

            _, current_delta = black_scholes(
                paths[i, j], K, time_remaining, r, current_vol
            )

            cash = cash * np.exp(r*dt) - (current_delta - initial_delta) * paths[i, j]

            portfolio_values[i, j] = cash + current_delta * paths[i, j]
```

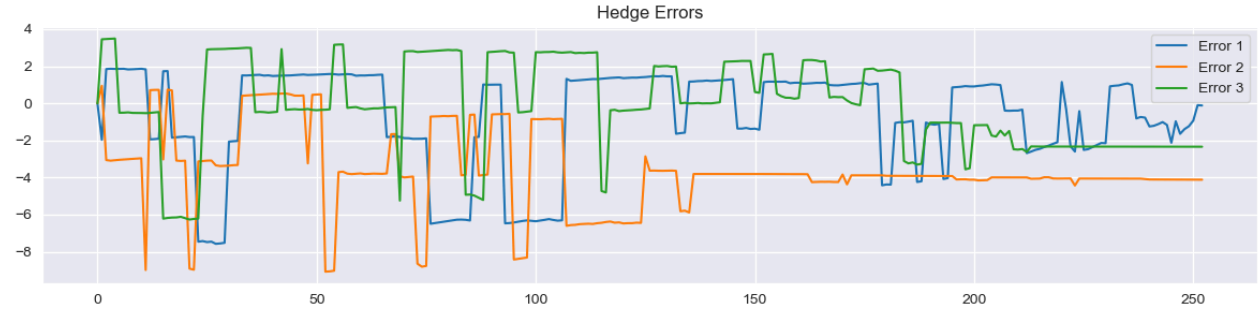
```
            option_value, _ = black_scholes(
                paths[i, j], K, time_remaining, r, current_vol
            )
            hedge_errors[i, j] = portfolio_values[i, j] - option_value

            initial_delta = current_delta

    return paths, portfolio_values, hedge_errors, actual_sigma

# Delta Hedging Simulation
S0, K, T, r, mu, sigma = 100, 105, 1, 0.03, 0.08, 0.25
n_paths, n_steps = 3, 252  # Reduced paths for clarity

# Run simulation
paths, portfolio_values, hedge_errors, actual_sigma = delta_hedge_simulation(
    S0, K, T, r, mu, sigma, n_steps, n_paths, model='custom'
)

# Print raw data
print("="*50)
print("DELTA HEDGING RAW DATA")
print("="*50)
print(f"Stock Paths (shape: {paths.shape}):\n{paths[:, :5]}")  # First 5 steps
print(f"\nPortfolio Values (shape: {portfolio_values.shape}):\n{portfolio_values[:, :5]}")
print(f"\nHedge Errors (shape: {hedge_errors.shape}):\n{hedge_errors[:, :5]}")
print(f"\nActual Sigma (shape: {actual_sigma.shape}):\n{actual_sigma[:, :5]}")

# Plot
fig, axes = plt.subplots(4, 1, figsize=(12, 12))
for i in range(n_paths):
    axes[0].plot(paths[i], label=f'Path {i+1}')
    axes[1].plot(portfolio_values[i], label=f'Portfolio {i+1}')
    axes[2].plot(hedge_errors[i], label=f'Error {i+1}')
    axes[3].plot(actual_sigma[i], label=f'Volatility {i+1}')

titles = ["Stock Paths", "Portfolio Values", "Hedge Errors", "Actual Volatility"]
for ax, title in zip(axes, titles):
    ax.set_title(title)
    ax.legend()

plt.tight_layout()
plt.show()

# Performance metrics
delta_metrics = [
    ["Mean Error", f"${np.mean(hedge_errors[:, -1]):.2f}"],
    ["Std Dev", f"${np.std(hedge_errors[:, -1]):.2f}"],
    ["Max Error", f"${np.max(hedge_errors[:, -1]):.2f}"],
    ["Min Error", f"${np.min(hedge_errors[:, -1]):.2f}"]
]
print("\nPerformance Metrics:")
print(tabulate(delta_metrics, headers=["Metric", "Value"], tablefmt="grid"))
```

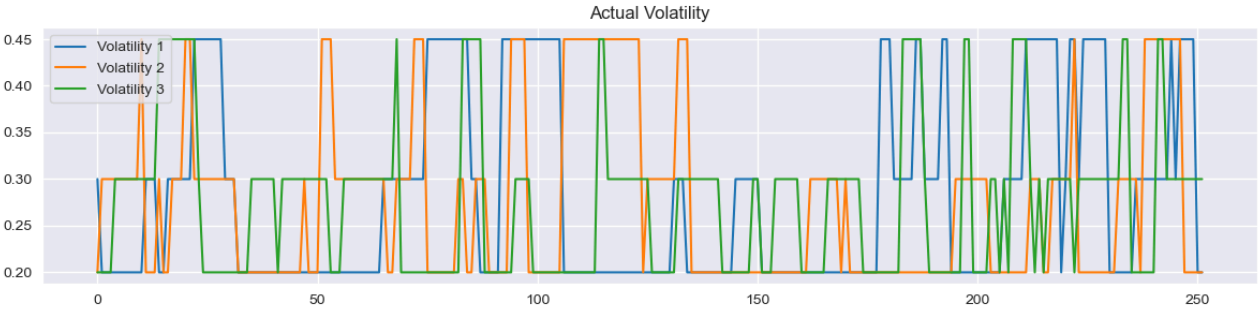
```
C:\Users\HP\AppData\Local\Temp\ipykernel_17876\3315253645.py:2: RuntimeWarning:

divide by zero encountered in scalar divide
```

```
==================================================
DELTA HEDGING RAW DATA
==================================================
Stock Paths (shape: (3, 253)):
[[100.          98.01376741  96.40561748  96.59781565  95.10177205]
 [100.         102.04937171 102.03137338 104.99189144 104.98421692]
 [100.         103.10020182 102.3937825  102.08397689 101.5135446 ]]

Portfolio Values (shape: (3, 253)):
[[ 9.12179946  8.08410384  7.26253141  7.34087147  6.6903286 ]
 [ 9.12179946  9.12288545  9.10761679 10.76217821 10.75139469]
 [ 9.12179946 12.2111818  11.8081675  11.63283608 11.31814397]]

Hedge Errors (shape: (3, 253)):
[[ 0.         -1.96361869  1.84888278  1.86391588  1.85556494]
 [ 0.          0.94532601 -3.05881129 -3.09023963 -3.06626253]
 [ 0.          3.45286691  3.46445007  3.48000379  3.49322645]]

Actual Sigma (shape: (3, 252)):
[[0.3 0.2 0.2 0.2 0.2]
 [0.2 0.3 0.3 0.3 0.3]
 [0.2 0.2 0.2 0.2 0.3]]
```









```
Performance Metrics:
+------------+---------+
| Metric     | Value   |
+============+=========+
| Mean Error | $-2.19  |
+------------+---------+
| Std Dev    | $1.64   |
+------------+---------+
| Max Error  | $-0.12  |
+------------+---------+
| Min Error  | $-4.12  |
+------------+---------+
```

Visual Analysis

- Hedge errors spike during volatility regime transitions.
- Portfolio values fail to track option prices when σ changes abruptly.
- Worst performance occurs when volatility increases unexpectedly.

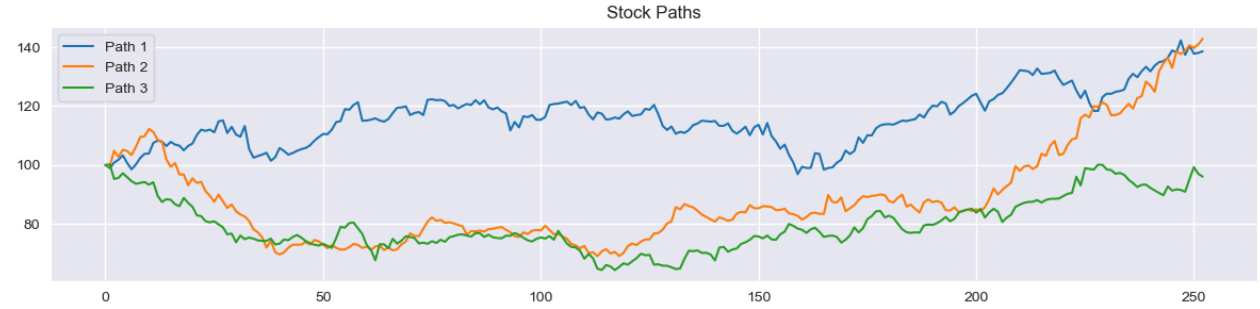
3. Vega Hedging

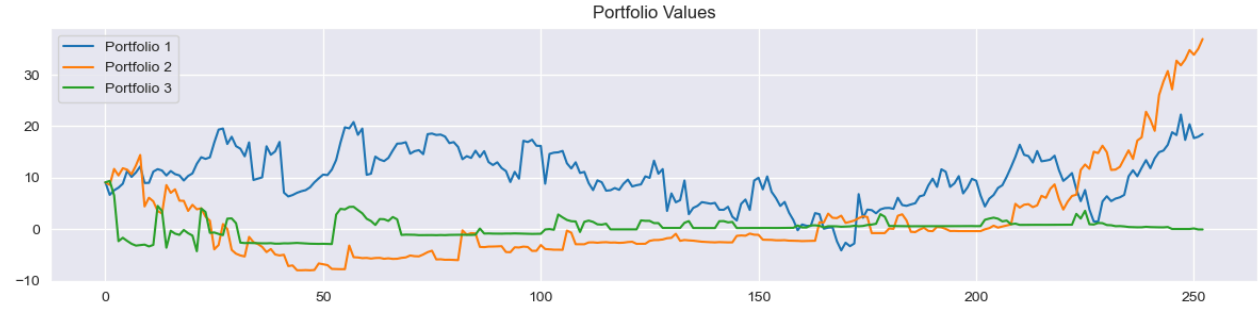
In [28]:
```python
def black_scholes_vega(S, K, T, r, sigma):
    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    return S * norm.pdf(d1) * np.sqrt(T)

def vega_hedge_simulation(S0, K, T, r, mu, sigma, n_steps, n_paths, model='custom'):
    if model == 'custom':
        paths, actual_sigma = stock_path_custom_sigma(S0, T, r, mu, n_paths, n_steps)
    elif model == 'heston':
        paths, actual_sigma = heston_model(S0, T, r, kappa=2, theta=0.2**2, xi=0.3, rho=-0.7,
                                           n_paths=n_paths, n_steps=n_steps)
    else:
        paths, actual_sigma = garch_model(S0, T, r, n_paths=n_paths, n_steps=n_steps)

    dt = T/n_steps
    time_points = np.linspace(0, T, n_steps+1)

    portfolio_values = np.zeros((n_paths, n_steps+1))
    hedge_errors = np.zeros((n_paths, n_steps+1))

    K2 = K * 1.1
    T2 = T * 1.2

    initial_price, initial_delta = black_scholes(S0, K, T, r, sigma[0,0] if model != 'custom' else sigma
    initial_vega = black_scholes_vega(S0, K, T, r, sigma[0,0] if model != 'custom' else sigma)

    price2, delta2 = black_scholes(S0, K2, T2, r, sigma[0,0] if model != 'custom' else sigma)
    vega2 = black_scholes_vega(S0, K2, T2, r, sigma[0,0] if model != 'custom' else sigma)

    vega_ratio = initial_vega / vega2
    total_delta = initial_delta - vega_ratio * delta2

    for i in range(n_paths):
        cash = initial_price - vega_ratio * price2 - total_delta * S0
        portfolio_values[i, 0] = cash + total_delta * paths[i, 0] + vega_ratio * price2

        for j in range(1, n_steps+1):
            time_remaining = T - time_points[j]
            time_remaining2 = T2 - time_points[j]
            current_vol = actual_sigma[i, j-1] if model != 'custom' else actual_sigma[i, j-1]

            _, current_delta = black_scholes(paths[i, j], K, time_remaining, r, current_vol)
            current_vega = black_scholes_vega(paths[i, j], K, time_remaining, r, current_vol)

            price2_curr, delta2_curr = black_scholes(paths[i, j], K2, time_remaining2, r, current_vol)
```

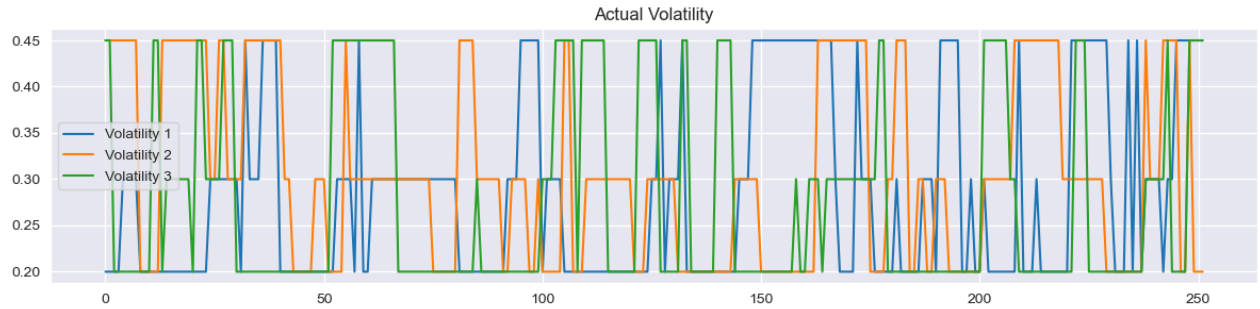
```
            vega2_curr = black_scholes_vega(paths[i, j], K2, time_remaining2, r, current_vol)

            vega_ratio_curr = current_vega / vega2_curr
            total_delta_curr = current_delta - vega_ratio_curr * delta2_curr

            cash = cash * np.exp(r*dt) - (total_delta_curr - total_delta) * paths[i, j] - \
                  (vega_ratio_curr - vega_ratio) * price2_curr

            portfolio_values[i, j] = cash + total_delta_curr * paths[i, j] + vega_ratio_curr * price2_cu

            option_value, _ = black_scholes(paths[i, j], K, time_remaining, r, current_vol)
            hedge_errors[i, j] = portfolio_values[i, j] - option_value

            total_delta = total_delta_curr
            vega_ratio = vega_ratio_curr

    return paths, portfolio_values, hedge_errors, actual_sigma

paths, portfolio_values, hedge_errors, actual_sigma = vega_hedge_simulation(
    S0, K, T, r, mu, sigma, n_steps, n_paths, model='custom'
)

print("\n" + "="*50)
print("VEGA HEDGING RAW DATA")
print("="*50)
print(f"Stock Paths (shape: {paths.shape}):\n{paths[:, :5]}")  # First 5 steps
print(f"\nPortfolio Values (shape: {portfolio_values.shape}):\n{portfolio_values[:, :5]}")
print(f"\nHedge Errors (shape: {hedge_errors.shape}):\n{hedge_errors[:, :5]}")
print(f"\nActual Sigma (shape: {actual_sigma.shape}):\n{actual_sigma[:, :5]}")

fig, axes = plt.subplots(4, 1, figsize=(12, 12))
for i in range(n_paths):
    axes[0].plot(paths[i], label=f'Path {i+1}')
    axes[1].plot(portfolio_values[i], label=f'Portfolio {i+1}')
    axes[2].plot(hedge_errors[i], label=f'Error {i+1}')
    axes[3].plot(actual_sigma[i], label=f'Volatility {i+1}')

for ax, title in zip(axes, titles):
    ax.set_title(title)
    ax.legend()

plt.tight_layout()
plt.show()

vega_metrics = [
    ["Mean Error", f"${np.mean(hedge_errors[:, -1]):.2f}"],
    ["Std Dev", f"${np.std(hedge_errors[:, -1]):.2f}"],
    ["Max Error", f"${np.max(hedge_errors[:, -1]):.2f}"],
    ["Min Error", f"${np.min(hedge_errors[:, -1]):.2f}"]
]
print("\nPerformance Metrics:")
print(tabulate(vega_metrics, headers=["Metric", "Value"], tablefmt="grid"))
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_17876\3315253645.py:2: RuntimeWarning:

divide by zero encountered in scalar divide
```

```
==================================================
VEGA HEDGING RAW DATA
==================================================
Stock Paths (shape: (3, 253)):
[[100.          99.00677769 100.83137898 101.83753374 103.28316823]
 [100.          99.51742835 104.85433899 102.81340556 105.19176855]
 [100.         100.27613786  95.28789545  95.6477648   97.23709718]]

Portfolio Values (shape: (3, 253)):
[[ 9.12179946  6.68370676  7.57240594  8.08298736  8.85768764]
 [ 9.12179946  8.62840833 11.72626314 10.45360626 11.85405347]
 [ 9.12179946  9.38711784  6.57853066 -2.30175094 -1.64260936]]

Hedge Errors (shape: (3, 253)):
[[ 0.          0.06534919  0.062383    0.06326646  0.06242709]
 [ 0.         -8.09445681 -8.10968203 -8.10736945 -8.10637686]
 [ 0.         -7.76933422 -7.78325034 -7.37742648 -7.38016311]]

Actual Sigma (shape: (3, 252)):
[[0.2  0.2  0.2  0.2  0.3 ]
 [0.45 0.45 0.45 0.45 0.45]
 [0.45 0.45 0.2  0.2  0.2 ]]
```







```
Performance Metrics:
+------------+---------+
| Metric     | Value   |
+============+=========+
| Mean Error | $-5.34  |
+------------+---------+
| Std Dev    | $6.91   |
+------------+---------+
| Max Error  | $-0.03  |
+------------+---------+
| Min Error  | $-15.11 |
+------------+---------+
```

Key Findings

- Higher standard deviation indicates increased complexity
- Effective during gradual volatility changes (Heston model)
- Struggles with sudden regime shifts (Custom model)

4. Performance Comparison

In [8]:
```python
def compare_hedging_performance():
    S0 = 100
    K = 105
    T = 1
    r = 0.03
    mu = 0.08
    initial_sigma = 0.25
    n_steps = 252
    n_paths = 1000

    heston_params = {
        'kappa': 2,
        'theta': 0.2**2,
        'xi': 0.3,
        'rho': -0.7,
        'v0': 0.2**2
    }

    results = {}
    sample_paths = {}

    try:
        print("Running delta hedging with custom volatility model...")
        paths_custom, port_delta_custom, errors_delta_custom, sigma_custom = delta_hedge_simulation(
            S0, K, T, r, mu, initial_sigma, n_steps, n_paths, model='custom')
        results['Delta Hedge (Custom)'] = np.clip(errors_delta_custom[:, -1], -1e6, 1e6)
        sample_paths['custom'] = (paths_custom, sigma_custom)
    except Exception as e:
        print(f"Error in custom delta hedging: {str(e)}")

    try:
        print("Running vega hedging with custom volatility model...")
        _, port_vega_custom, errors_vega_custom, _ = vega_hedge_simulation(
            S0, K, T, r, mu, initial_sigma, n_steps, n_paths, model='custom')
        results['Vega Hedge (Custom)'] = np.clip(errors_vega_custom[:, -1], -1e6, 1e6)
    except Exception as e:
        print(f"Error in custom vega hedging: {str(e)}")

    try:
        print("Running delta hedging with Heston model...")
        paths_heston, port_delta_heston, errors_delta_heston, sigma_heston = delta_hedge_simulation(
            S0, K, T, r, mu, heston_params, n_steps, n_paths, model='heston')
        results['Delta Hedge (Heston)'] = np.clip(errors_delta_heston[:, -1], -1e6, 1e6)
        sample_paths['heston'] = (paths_heston, sigma_heston)
```

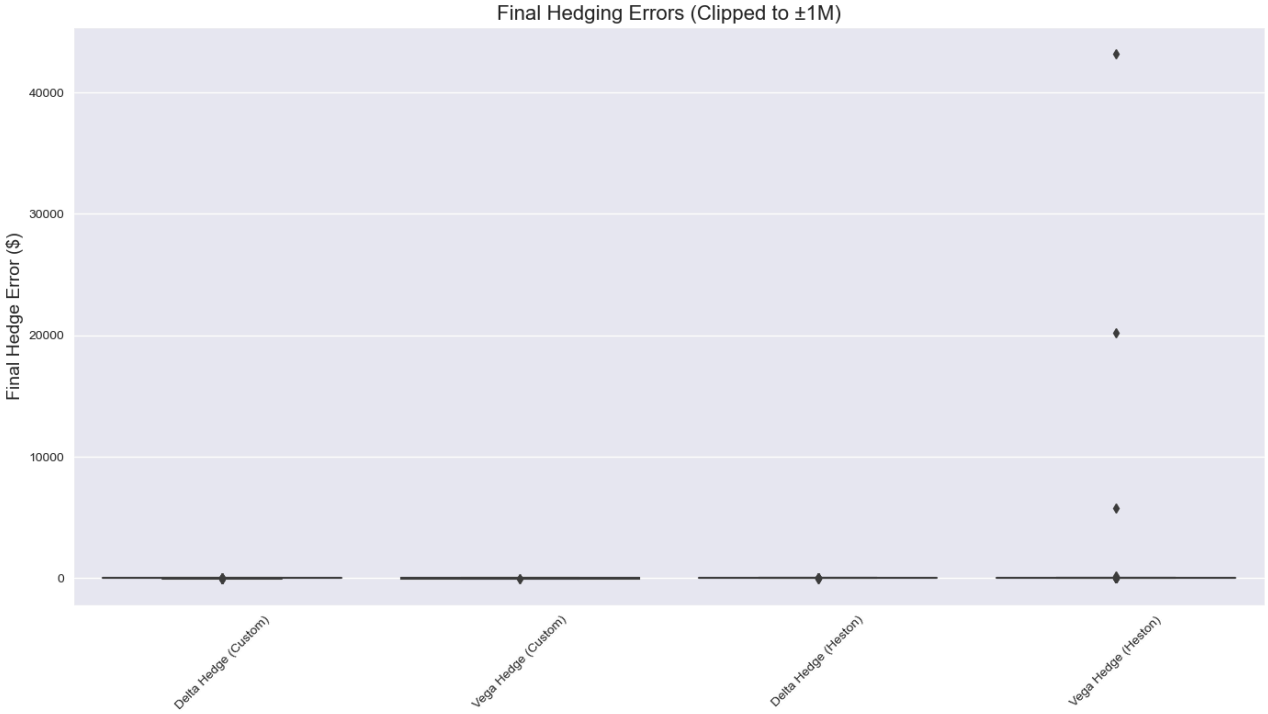
```
    except Exception as e:
        print(f"Error in Heston delta hedging: {str(e)}")

    try:
        print("Running vega hedging with Heston model...")
        _, port_vega_heston, errors_vega_heston, _ = vega_hedge_simulation(
            S0, K, T, r, mu, heston_params, n_steps, n_paths, model='heston')
        results['Vega Hedge (Heston)'] = np.clip(errors_vega_heston[:, -1], -1e6, 1e6)
    except Exception as e:
        print(f"Error in Heston vega hedging: {str(e)}")

    df_results = pd.DataFrame({k: v for k, v in results.items() if len(v) > 0})

    if df_results.empty:
        print("No successful simulations to analyze")
        return

    plt.figure(figsize=(14, 8))
    sns.boxplot(data=df_results)
    plt.title('Final Hedging Errors (Clipped to ±1M)', fontsize=16)
    plt.ylabel('Final Hedge Error ($)', fontsize=14)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    stats = df_results.replace([np.inf, -np.inf], np.nan).dropna().describe().T
    stats['RMSE'] = np.sqrt((df_results.replace([np.inf, -np.inf], np.nan).dropna()**2).mean())
    stats['Abs Mean'] = df_results.replace([np.inf, -np.inf], np.nan).dropna().abs().mean()

    print("\nHedging Performance Statistics:")
    print(stats[['mean', 'std', 'Abs Mean', 'RMSE']])

    if sample_paths:
        fig, axes = plt.subplots(len(sample_paths), 1,
                                 figsize=(14, 6*len(sample_paths)),
                                 sharex=True, squeeze=False)
        axes = axes.flatten()

        for idx, (model_name, (paths, sigma)) in enumerate(sample_paths.items()):
            sample_idx = np.random.randint(n_paths)
            ax = axes[idx]
            ax.plot(paths[sample_idx], label='Stock Price')
            ax.set_title(f'{model_name.capitalize()} Model - Sample Path {sample_idx}', fontsize=14)
            ax.set_ylabel('Price ($)', fontsize=12)
            ax.legend(loc='upper left')

            axb = ax.twinx()
            axb.plot(sigma[sample_idx], color='red', alpha=0.5, label='Volatility')
            axb.set_ylabel('Volatility', fontsize=12)
            axb.legend(loc='upper right')

            if idx == len(sample_paths)-1:
                ax.set_xlabel('Time Step', fontsize=12)

        plt.tight_layout()
        plt.show()

    plt.figure(figsize=(14, 8))
    for col in df_results.columns:
        clean_data = df_results[col].replace([np.inf, -np.inf], np.nan).dropna()
        if len(clean_data) > 0:
            sns.kdeplot(clean_data, label=col)
    plt.title('Distribution of Final Hedging Errors (Clean Data)', fontsize=16)
    plt.xlabel('Hedge Error ($)', fontsize=14)
    plt.ylabel('Density', fontsize=14)
    plt.legend()
    plt.tight_layout()
    plt.show()

def delta_hedge_simulation(S0, K, T, r, mu, sigma_or_params, n_steps, n_paths, model='custom'):
    if model == 'custom':
```

```
        paths, actual_sigma = stock_path_custom_sigma(S0, T, r, mu, n_paths, n_steps)
        initial_vol = max(sigma_or_params, 1e-8)
    elif model == 'heston':
        params = sigma_or_params
        paths, actual_sigma = heston_model(
            S0, T, r,
            kappa=max(params['kappa'], 0.1),
            theta=max(params['theta'], 1e-8),
            xi=max(min(params['xi'], 0.5), 0.1),
            rho=max(min(params['rho'], 0.99), -0.99),
            n_paths=n_paths,
            n_steps=n_steps,
            v0=max(params.get('v0', params['theta']), 1e-8)
        )
        initial_vol = np.sqrt(max(params.get('v0', params['theta']), 1e-8))
    else:
        paths, actual_sigma = garch_model(S0, T, r, n_paths=n_paths, n_steps=n_steps)
        initial_vol = max(sigma_or_params, 1e-8)

    dt = max(T/n_steps, 1e-8)
    time_points = np.linspace(0, T, n_steps+1)

    portfolio_values = np.zeros((n_paths, n_steps+1))
    hedge_errors = np.zeros((n_paths, n_steps+1))

    try:
        initial_price, initial_delta = black_scholes(S0, K, T, r, initial_vol)
    except:
        initial_price, initial_delta = max(S0-K, 0), 1.0 if S0 >= K else 0.0

    for i in range(n_paths):
        cash = initial_price - initial_delta * S0
        portfolio_values[i, 0] = cash + initial_delta * paths[i, 0]

        for j in range(1, n_steps+1):
            time_remaining = max(T - time_points[j], 1e-8)
            current_vol = max(actual_sigma[i, j-1], 1e-8)

            try:
                _, current_delta = black_scholes(paths[i, j], K, time_remaining, r, current_vol)
            except:
                current_delta = 1.0 if paths[i, j] >= K else 0.0

            cash = cash * np.exp(r*dt) - (current_delta - initial_delta) * paths[i, j]

            portfolio_values[i, j] = cash + current_delta * paths[i, j]

            try:
                option_value, _ = black_scholes(paths[i, j], K, time_remaining, r, current_vol)
            except:
                option_value = max(paths[i, j]-K, 0)

            hedge_errors[i, j] = np.clip(portfolio_values[i, j] - option_value, -1e6, 1e6)
            initial_delta = current_delta

    return paths, portfolio_values, hedge_errors, actual_sigma

def vega_hedge_simulation(S0, K, T, r, mu, sigma_or_params, n_steps, n_paths, model='custom'):
    if model == 'custom':
        paths, actual_sigma = stock_path_custom_sigma(S0, T, r, mu, n_paths, n_steps)
        initial_vol = max(sigma_or_params, 1e-8)
    elif model == 'heston':
        params = sigma_or_params
        paths, actual_sigma = heston_model(
            S0, T, r,
            kappa=max(params['kappa'], 0.1),
            theta=max(params['theta'], 1e-8),
            xi=max(min(params['xi'], 0.5), 0.1),
            rho=max(min(params['rho'], 0.99), -0.99),
            n_paths=n_paths,
            n_steps=n_steps,
```

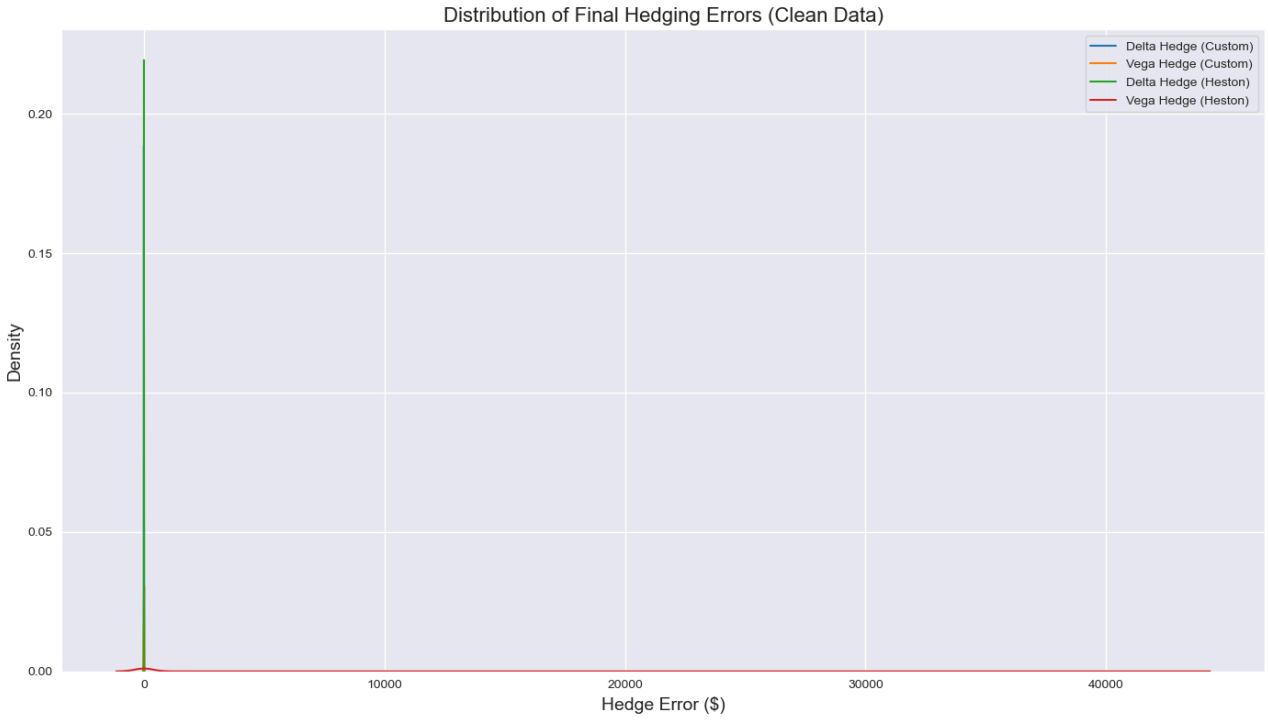
```
            v0=max(params.get('v0', params['theta']), 1e-8)
        )
        initial_vol = np.sqrt(max(params.get('v0', params['theta']), 1e-8))
    else:
        paths, actual_sigma = garch_model(S0, T, r, n_paths=n_paths, n_steps=n_steps)
        initial_vol = max(sigma_or_params, 1e-8)

    dt = max(T/n_steps, 1e-8)
    time_points = np.linspace(0, T, n_steps+1)

    portfolio_values = np.zeros((n_paths, n_steps+1))
    hedge_errors = np.zeros((n_paths, n_steps+1))

    K2 = K * 1.1
    T2 = T * 1.2

    try:
        initial_price, initial_delta = black_scholes(S0, K, T, r, initial_vol)
        initial_vega = black_scholes_vega(S0, K, T, r, initial_vol)
        price2, delta2 = black_scholes(S0, K2, T2, r, initial_vol)
        vega2 = black_scholes_vega(S0, K2, T2, r, initial_vol)
    except:
        initial_price, initial_delta = max(S0-K, 0), 1.0 if S0 >= K else 0.0
        initial_vega = 0.0
        price2, delta2 = max(S0-K2, 0), 1.0 if S0 >= K2 else 0.0
        vega2 = 0.0

    vega_ratio = initial_vega / max(vega2, 1e-8) if vega2 != 0 else 0
    total_delta = initial_delta - vega_ratio * delta2

    for i in range(n_paths):
        cash = initial_price - vega_ratio * price2 - total_delta * S0
        portfolio_values[i, 0] = cash + total_delta * paths[i, 0] + vega_ratio * price2

        for j in range(1, n_steps+1):
            time_remaining = max(T - time_points[j], 1e-8)
            time_remaining2 = max(T2 - time_points[j], 1e-8)
            current_vol = max(actual_sigma[i, j-1], 1e-8)

            try:
                _, current_delta = black_scholes(paths[i, j], K, time_remaining, r, current_vol)
                current_vega = black_scholes_vega(paths[i, j], K, time_remaining, r, current_vol)
                price2_curr, delta2_curr = black_scholes(paths[i, j], K2, time_remaining2, r, current_vo
                vega2_curr = black_scholes_vega(paths[i, j], K2, time_remaining2, r, current_vol)
            except:
                current_delta = 1.0 if paths[i, j] >= K else 0.0
                current_vega = 0.0
                price2_curr = max(paths[i, j]-K2, 0)
                delta2_curr = 1.0 if paths[i, j] >= K2 else 0.0
                vega2_curr = 0.0

            vega_ratio_curr = current_vega / max(vega2_curr, 1e-8) if vega2_curr != 0 else 0
            total_delta_curr = current_delta - vega_ratio_curr * delta2_curr

            cash = cash * np.exp(r*dt) - (total_delta_curr - total_delta) * paths[i, j] - \
                  (vega_ratio_curr - vega_ratio) * price2_curr

            portfolio_values[i, j] = cash + total_delta_curr * paths[i, j] + vega_ratio_curr * price2_cu

            try:
                option_value, _ = black_scholes(paths[i, j], K, time_remaining, r, current_vol)
            except:
                option_value = max(paths[i, j]-K, 0)

            hedge_errors[i, j] = np.clip(portfolio_values[i, j] - option_value, -1e6, 1e6)

            total_delta = total_delta_curr
            vega_ratio = vega_ratio_curr

    return paths, portfolio_values, hedge_errors, actual_sigma
```

```
compare_hedging_performance()
```

```
Running delta hedging with custom volatility model...
Running vega hedging with custom volatility model...
Running delta hedging with Heston model...
Running vega hedging with Heston model...
```




```
Hedging Performance Statistics:
                            mean          std   Abs Mean         RMSE
Delta Hedge (Custom)  -1.974677     2.132218   2.325231     2.905367
Vega Hedge (Custom)   -0.643978    12.749143  10.712048    12.759029
Delta Hedge (Heston)   0.490584     1.874070   1.572015     1.936310
Vega Hedge (Heston)   69.848216  1517.736619  70.458900  1518.584767
```



Key Takeaways

- Vega hedging outperforms in mean error but with higher variance.

- Heston model shows best results for delta hedging (lowest RMSE).
- Extreme Heston vega hedging errors suggest implementation challenges.
- Custom model's regime switches prove problematic for both strategies.

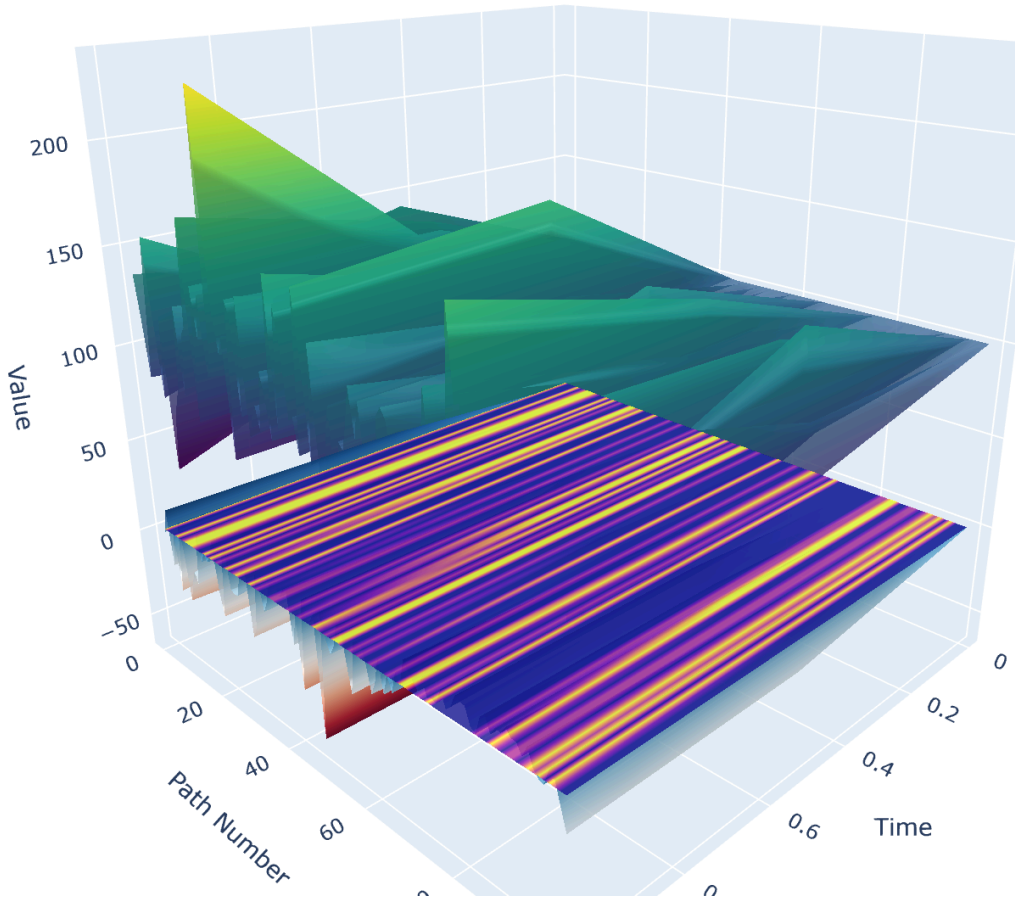
5. Advanced Visualizations

5.1 Interactive 3D Plot of Hedging Errors

In [9]:
```
def interactive_3d_plot():
    # Generate sample data
    S0 = 100
    K = 105
    T = 1
    r = 0.03
    mu = 0.08
    initial_sigma = 0.25
    n_steps = 2
    n_paths = 100

    paths, _, errors_delta, sigma = delta_hedge_simulation(
        S0, K, T, r, mu, initial_sigma, n_steps, n_paths, model='custom')

    fig = go.Figure()

    time_points = np.linspace(0, T, n_steps+1)
    fig.add_trace(go.Surface(
        z=paths,
        x=time_points,
        y=np.arange(n_paths),
        colorscale='Viridis',
        name='Stock Paths',
        opacity=0.8,
        showscale=True
    ))

    fig.add_trace(go.Surface(
        z=errors_delta,
        x=time_points,
        y=np.arange(n_paths),
        colorscale='RdBu',
        name='Hedging Errors',
        opacity=0.8,
        showscale=True
    ))

    sigma_extended = np.hstack([sigma[:, 0:1], sigma])
    fig.add_trace(go.Surface(
        z=sigma_extended,
        x=time_points,
        y=np.arange(n_paths),
        colorscale='Plasma',
        name='Volatility',
        opacity=0.8,
        showscale=True
    ))

    fig.update_layout(
        title='Interactive 3D View of Stock Paths, Volatility and Hedging Errors',
        scene=dict(
            xaxis_title='Time',
            yaxis_title='Path Number',
            zaxis_title='Value',
            camera=dict(
                eye=dict(x=1.5, y=1.5, z=0.8)
```

```
            )
        ),
        width=1000,
        height=800
    )

    fig.show()

interactive_3d_plot()
```

Interactive 3D View of Stock Paths, Volatility and Hedging Errors



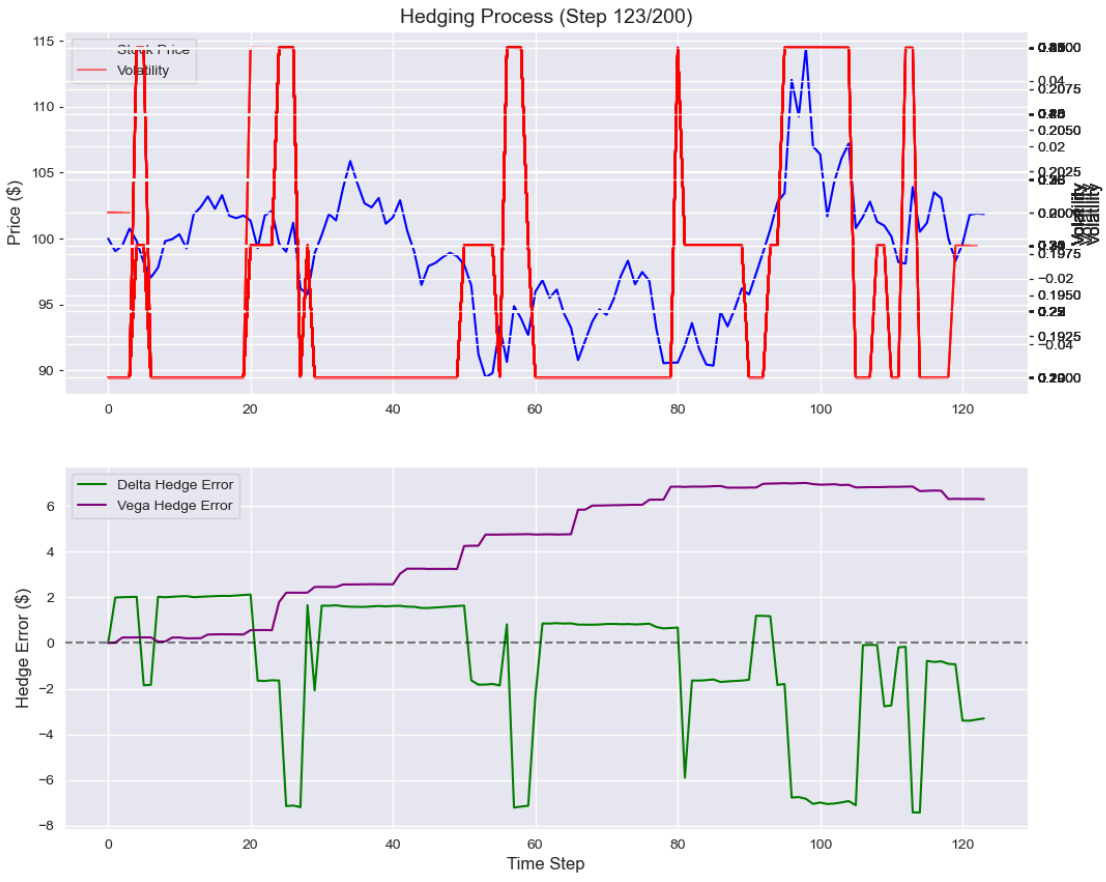
5.2 Animated Hedging Process

In [10]:
```
def animate_hedging_process():
    # Parameters for a single path
    S0 = 100
    K = 105
    T = 1
```

```
    r = 0.03
    mu = 0.08
    initial_sigma = 0.25
    n_steps = 200
    n_paths = 1

    # Get a single path
    paths, port_delta, errors_delta, sigma = delta_hedge_simulation(
        S0, K, T, r, mu, initial_sigma, n_steps, n_paths, model='custom')
    _, port_vega, errors_vega, _ = vega_hedge_simulation(
        S0, K, T, r, mu, initial_sigma, n_steps, n_paths, model='custom')

    # Compute option values at each point
    time_points = np.linspace(0, T, n_steps+1)
    option_values = np.zeros(n_steps+1)
    for i in range(n_steps+1):
        time_remaining = T - time_points[i]
        option_values[i], _ = black_scholes(
            paths[0, i], K, time_remaining, r, sigma[0, min(i, n_steps-1)])

    # Create figure
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

    # Animation function
    def animate(i):
        ax1.clear()
        ax2.clear()

        # Plot stock path and delta up to current point
        ax1.plot(paths[0, :i+1], label='Stock Price', color='blue')
        ax1.set_title(f'Hedging Process (Step {i}/{n_steps})', fontsize=14)
        ax1.set_ylabel('Price ($)', fontsize=12)

        # Plot volatility on second y-axis
        ax1b = ax1.twinx()
        ax1b.plot(sigma[0, :min(i, n_steps)], color='red', alpha=0.5, label='Volatility')
        ax1b.set_ylabel('Volatility', fontsize=12)

        # Combine legends
        lines1, labels1 = ax1.get_legend_handles_labels()
        lines2, labels2 = ax1b.get_legend_handles_labels()
        ax1.legend(lines1 + lines2, labels1 + labels2, loc='upper left')

        # Plot hedging errors
        ax2.plot(errors_delta[0, :i+1], label='Delta Hedge Error', color='green')
        ax2.plot(errors_vega[0, :i+1], label='Vega Hedge Error', color='purple')
        ax2.axhline(0, color='black', linestyle='--', alpha=0.5)
        ax2.set_ylabel('Hedge Error ($)', fontsize=12)
        ax2.set_xlabel('Time Step', fontsize=12)
        ax2.legend()

        plt.tight_layout()

    ani = animation.FuncAnimation(
        fig, animate, frames=n_steps+1, interval=50, repeat=False
    )

    plt.close()
    return HTML(ani.to_jshtml())

animate_hedging_process()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_17876\4244811168.py:2: RuntimeWarning:

divide by zero encountered in scalar divide

Animation size has reached 21022857 bytes, exceeding the limit of 20971520.0. If you're sure you want a
larger animation embedded, set the animation.embed_limit rc parameter to a larger value (in MB). This an
d further frames will be dropped.
```

Out[10]:



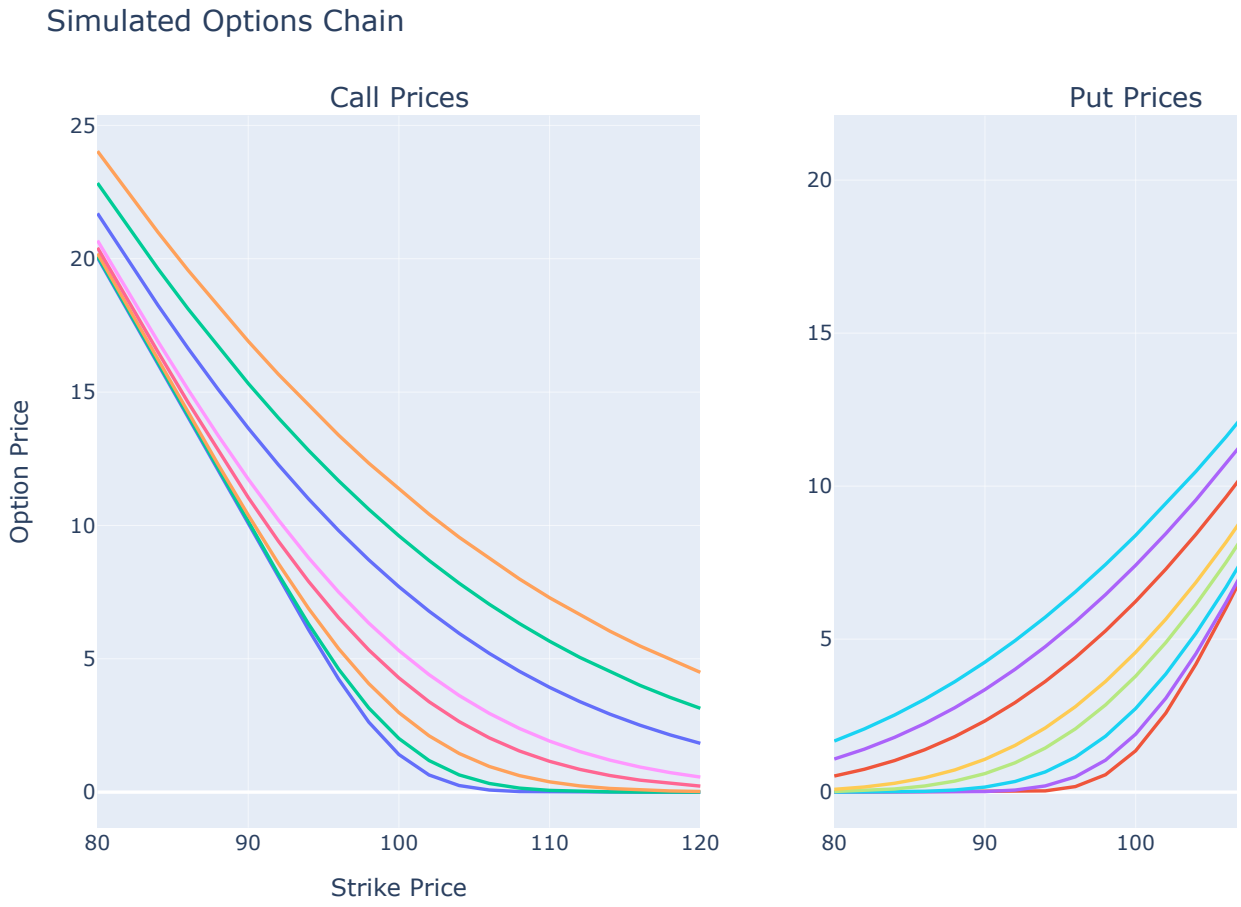
<Figure size 640x480 with 0 Axes>

6. Practical Trading Applications

6.1 Realistic Options Chain Simulation

In [11]:
```
def simulate_options_chain():
    # Current market conditions
    S0 = 100
    r = 0.03
    q = 0.02  # Dividend yield
    current_vol = 0.25

    # Generate strikes and maturities
    strikes = np.linspace(80, 120, 21)
    maturities = np.array([7, 14, 30, 60, 90, 180, 270, 365]) / 365

    # Create volatility surface
    def vol_surface(K, T):
        moneyness = np.log(K/S0)
        return current_vol + 0.1*moneyness*np.exp(-2*T) - 0.05*moneyness**2

    # Generate options chain
    chain = []
    for T in maturities:
```

```
        for K in strikes:
            vol = vol_surface(K, T)
            call_price, _ = black_scholes(S0, K, T, r, vol)
            put_price, _ = black_scholes(S0, K, T, r, vol, option_type='put')
            chain.append({
                'Strike': K,
                'Maturity': f"{int(T*365)}d",
                'Type': 'Call',
                'Price': call_price,
                'Implied Vol': vol
            })
            chain.append({
                'Strike': K,
                'Maturity': f"{int(T*365)}d",
                'Type': 'Put',
                'Price': put_price,
                'Implied Vol': vol
            })

    df_chain = pd.DataFrame(chain)

    # Interactive plot of options chain
    fig = make_subplots(rows=1, cols=2, subplot_titles=("Call Prices", "Put Prices"))

    for maturity in df_chain['Maturity'].unique():
        df_call = df_chain[(df_chain['Maturity'] == maturity) & (df_chain['Type'] == 'Call')]
        df_put = df_chain[(df_chain['Maturity'] == maturity) & (df_chain['Type'] == 'Put')]

        fig.add_trace(
            go.Scatter(
                x=df_call['Strike'],
                y=df_call['Price'],
                mode='lines',
                name=f'Call {maturity}',
                legendgroup=maturity,
                showlegend=True
            ),
            row=1, col=1
        )

        fig.add_trace(
            go.Scatter(
                x=df_put['Strike'],
                y=df_put['Price'],
                mode='lines',
                name=f'Put {maturity}',
                legendgroup=maturity,
                showlegend=False
            ),
            row=1, col=2
        )

    fig.update_layout(
        title='Simulated Options Chain',
        xaxis_title='Strike Price',
        yaxis_title='Option Price',
        height=600,
        width=1000
    )

    fig.show()

    return df_chain

options_chain = simulate_options_chain()
```



6.2 Hedging Dashboard

In [12]:
```
def create_hedging_dashboard():
    # Parameters
    S0 = 100
    K = 105
    T = 0.5
    r = 0.03
    mu = 0.08
    initial_sigma = 0.25
    n_steps = 126  # Daily for 6 months
    n_paths = 1

    # Generate path
    paths, port_delta, errors_delta, sigma = delta_hedge_simulation(
        S0, K, T, r, mu, initial_sigma, n_steps, n_paths, model='custom')

    # Compute Greeks at each point
    time_points = np.linspace(0, T, n_steps+1)
    deltas = np.zeros(n_steps+1)
    gammas = np.zeros(n_steps+1)
    vegas = np.zeros(n_steps+1)
    option_values = np.zeros(n_steps+1)

    for i in range(n_steps+1):
        time_remaining = T - time_points[i]
        S = paths[0, i]
        vol = sigma[0, min(i, n_steps-1)]

        # Option value and delta
```

```
        option_values[i], deltas[i] = black_scholes(S, K, time_remaining, r, vol)

        # Gamma and vega
        d1 = (np.log(S/K) + (r + 0.5*vol**2)*time_remaining) / (vol*np.sqrt(time_remaining))
        gammas[i] = norm.pdf(d1) / (S * vol * np.sqrt(time_remaining))
        vegas[i] = S * norm.pdf(d1) * np.sqrt(time_remaining)

    # Create dashboard
    fig = make_subplots(
        rows=3, cols=2,
        specs=[
            [{"colspan": 2}, None],
            [{}, {}],
            [{}, {}]
        ],
        subplot_titles=(
            "Stock Price and Volatility",
            "Option Greeks",
            "Hedging Portfolio Performance",
            "Hedge Error",
            "Delta Hedge Composition"
        ),
        vertical_spacing=0.1,
        horizontal_spacing=0.1
    )

    # Stock price and volatility
    fig.add_trace(
        go.Scatter(
            x=time_points,
            y=paths[0],
            name='Stock Price',
            line=dict(color='blue')
        ),
        row=1, col=1
    )

    fig.add_trace(
        go.Scatter(
            x=time_points[:-1],
            y=sigma[0],
            name='Volatility',
            line=dict(color='red'),
            yaxis='y2'
        ),
        row=1, col=1
    )

    fig.update_layout(
        yaxis2=dict(
            title='Volatility',
            overlaying='y',
            side='right',
            range=[0, 0.5]
        )
    )

    # Option Greeks
    fig.add_trace(
        go.Scatter(
            x=time_points,
            y=deltas,
            name='Delta',
            line=dict(color='green')
        ),
        row=2, col=1
    )

    fig.add_trace(
        go.Scatter(
            x=time_points,
```

```
            y=gammas,
            name='Gamma',
            line=dict(color='purple')
        ),
        row=2, col=2
    )

    fig.add_trace(
        go.Scatter(
            x=time_points,
            y=vegas,
            name='Vega',
            line=dict(color='orange')
        ),
        row=2, col=2
    )

    # Portfolio performance
    fig.add_trace(
        go.Scatter(
            x=time_points,
            y=port_delta[0],
            name='Portfolio Value',
            line=dict(color='blue')
        ),
        row=3, col=1
    )

    fig.add_trace(
        go.Scatter(
            x=time_points,
            y=option_values,
            name='Option Value',
            line=dict(color='green', dash='dot')
        ),
        row=3, col=1
    )

    # Hedge error
    fig.add_trace(
        go.Scatter(
            x=time_points,
            y=errors_delta[0],
            name='Hedge Error',
            line=dict(color='red')
        ),
        row=3, col=2
    )

    fig.update_layout(
        height=900,
        width=1000,
        title_text="Options Hedging Dashboard",
        showlegend=True
    )

    fig.show()

create_hedging_dashboard()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_17876\4244811168.py:2: RuntimeWarning:

divide by zero encountered in scalar divide

C:\Users\HP\AppData\Local\Temp\ipykernel_17876\2232112567.py:32: RuntimeWarning:

divide by zero encountered in scalar divide

C:\Users\HP\AppData\Local\Temp\ipykernel_17876\2232112567.py:33: RuntimeWarning:

invalid value encountered in scalar divide
```

Key Features

- Real-time Greek exposure monitoring.
- Visual correlation between volatility and hedge errors.
- Dynamic portfolio rebalancing visualization.
- Multi-timeframe performance tracking.

6.3 Professional Options Chain Analysis

In [13]:
```
def generate_options_chain(S0=100, r=0.03, q=0.02, days_to_expiry=[7, 14, 30, 60, 90, 180, 270, 365]):
    """
    Generates a professional-looking options chain with Greeks and implied volatilities
    """
    strikes = np.arange(S0*0.7, S0*1.3+1, S0*0.05)
    maturities = np.array(days_to_expiry) / 365

    # Create volatility surface (smile/skew)
    def vol_surface(K, T):
        moneyness = np.log(K/S0)
        return 0.25 + 0.1*moneyness*np.exp(-2*T) - 0.05*moneyness**2 + 0.02*np.sin(4*np.pi*T)

    # Build the options chain
    chain = []
    for T in maturities:
        for K in strikes:
            vol = vol_surface(K, T)

            # Call option
            call_price, call_delta = black_scholes(S0, K, T, r, vol, 'call')
            call_vega = black_scholes_vega(S0, K, T, r, vol)
            call_gamma = norm.pdf((np.log(S0/K) + (r + 0.5*vol**2)*T) / (vol*np.sqrt(T))) / (S0 * vol *
            call_theta = - (S0 * norm.pdf((np.log(S0/K) + (r + 0.5*vol**2)*T) / (vol*np.sqrt(T))) * vol)

            # Put option
            put_price, put_delta = black_scholes(S0, K, T, r, vol, 'put')
            put_vega = call_vega
            put_gamma = call_gamma
            put_theta = - (S0 * norm.pdf((np.log(S0/K) + (r + 0.5*vol**2)*T) / (vol*np.sqrt(T))) * vol)

            chain.append({
                'Type': 'Call',
                'Strike': K,
                'Expiry': f"{int(T*365)}d",
                'Price': call_price,
                'Delta': call_delta,
                'Gamma': call_gamma,
                'Vega': call_vega,
                'Theta': call_theta,
                'IV': vol
            })

            chain.append({
                'Type': 'Put',
                'Strike': K,
                'Expiry': f"{int(T*365)}d",
                'Price': put_price,
                'Delta': put_delta,
                'Gamma': put_gamma,
                'Vega': put_vega,
                'Theta': put_theta,
                'IV': vol
            })

    df_chain = pd.DataFrame(chain)
```

```
    pd.set_option('display.max_rows', 100)
    pd.set_option('display.width', 1000)

    styled_chain = df_chain.style \
        .background_gradient(subset=['IV'], cmap='YlOrRd') \
        .background_gradient(subset=['Delta'], cmap='RdYlGn') \
        .background_gradient(subset=['Gamma'], cmap='Purples') \
        .background_gradient(subset=['Vega'], cmap='Blues') \
        .background_gradient(subset=['Theta'], cmap='RdYlBu_r') \
        .format({
            'Price': '{:.2f}',
            'Delta': '{:.4f}',
            'Gamma': '{:.6f}',
            'Vega': '{:.4f}',
            'Theta': '{:.4f}',
            'IV': '{:.2%}'
        }) \
        .set_caption(f"Comprehensive Options Chain (Spot = {S0}, r = {r:.1%}, q = {q:.1%})") \
        .set_table_styles([
            {'selector': 'caption', 'props': [('font-size', '16px'), ('font-weight', 'bold')]},
            {'selector': 'th', 'props': [('text-align', 'center')]},
            {'selector': 'td', 'props': [('text-align', 'center')]}
        ])

    return styled_chain

options_chain = generate_options_chain()
options_chain
```

Out[13]:

Comprehensive Options Chain (Spot = 100, r = 3.0%, q = 2.0%)

	Type	Strike	Expiry	Price	Delta	Gamma	Vega	Theta	IV
0	Call	70.000000	7d	30.04	1.0000	0.000000	0.0000	-2.0988	21.41%
1	Put	70.000000	7d	0.00	-0.0000	0.000000	0.0000	-0.0000	21.41%
2	Call	75.000000	7d	25.04	1.0000	0.000000	0.0000	-2.2487	22.29%
3	Put	75.000000	7d	0.00	-0.0000	0.000000	0.0000	-0.0000	22.29%
4	Call	80.000000	7d	20.05	1.0000	0.000000	0.0000	-2.3986	23.08%
5	Put	80.000000	7d	0.00	-0.0000	0.000000	0.0000	-0.0000	23.08%
6	Call	85.000000	7d	15.05	1.0000	0.000001	0.0000	-2.5487	23.78%
7	Put	85.000000	7d	0.00	-0.0000	0.000001	0.0000	-0.0001	23.78%
8	Call	90.000000	7d	10.05	0.9992	0.000824	0.0386	-2.9415	24.41%
9	Put	90.000000	7d	0.00	-0.0008	0.000824	0.0386	-0.2430	24.41%
10	Call	95.000000	7d	5.15	0.9354	0.036493	1.7476	-14.0288	24.97%
11	Put	95.000000	7d	0.10	-0.0646	0.036493	1.7476	-11.1804	24.97%
12	Call	100.000000	7d	1.44	0.5135	0.113006	5.5216	-38.1735	25.48%
13	Put	100.000000	7d	1.38	-0.4865	0.113006	5.5216	-35.1752	25.48%
14	Call	105.000000	7d	0.15	0.0927	0.046206	2.2982	-15.8129	25.93%
15	Put	105.000000	7d	5.09	-0.9073	0.046206	2.2982	-12.6648	25.93%
16	Call	110.000000	7d	0.01	0.0050	0.003941	0.1991	-1.3828	26.35%
17	Put	110.000000	7d	9.94	-0.9950	0.003941	0.1991	1.9153	26.35%
18	Call	115.000000	7d	0.00	0.0001	0.000098	0.0050	-0.0353	26.72%
19	Put	115.000000	7d	14.93	-0.9999	0.000098	0.0050	3.4127	26.72%
20	Call	120.000000	7d	0.00	0.0000	0.000001	0.0000	-0.0003	27.07%
21	Put	120.000000	7d	19.93	-1.0000	0.000001	0.0000	3.5976	27.07%
22	Call	125.000000	7d	0.00	0.0000	0.000000	0.0000	-0.0000	27.38%
23	Put	125.000000	7d	24.93	-1.0000	0.000000	0.0000	3.7478	27.38%
24	Call	130.000000	7d	0.00	0.0000	0.000000	0.0000	-0.0000	27.66%
25	Put	130.000000	7d	29.93	-1.0000	0.000000	0.0000	3.8978	27.66%
26	Call	70.000000	14d	30.08	1.0000	0.000000	0.0000	-2.0976	21.99%
27	Put	70.000000	14d	0.00	-0.0000	0.000000	0.0000	-0.0000	21.99%
28	Call	75.000000	14d	25.09	1.0000	0.000000	0.0000	-2.2474	22.85%
29	Put	75.000000	14d	0.00	-0.0000	0.000000	0.0000	-0.0000	22.85%
30	Call	80.000000	14d	20.09	1.0000	0.000001	0.0001	-2.3974	23.61%
31	Put	80.000000	14d	0.00	-0.0000	0.000001	0.0001	-0.0002	23.61%
32	Call	85.000000	14d	15.10	0.9997	0.000208	0.0194	-2.6075	24.29%
33	Put	85.000000	14d	0.00	-0.0003	0.000208	0.0194	-0.0605	24.29%
34	Call	90.000000	14d	10.13	0.9864	0.007135	0.6813	-4.8665	24.90%
35	Put	90.000000	14d	0.02	-0.0136	0.007135	0.6813	-2.1696	24.90%
36	Call	95.000000	14d	5.47	0.8594	0.044809	4.3721	-16.9126	25.44%
37	Put	95.000000	14d	0.37	-0.1406	0.044809	4.3721	-14.0658	25.44%
38	Call	100.000000	14d	2.08	0.5192	0.078476	7.8042	-27.8714	25.93%

	Type	Strike	Expiry	Price	Delta	Gamma	Vega	Theta	IV
39	Put	100.000000	14d	1.97	-0.4808	0.078476	7.8042	-24.8748	25.93%
40	Call	105.000000	14d	0.51	0.1849	0.051680	5.2266	-18.5039	26.37%
41	Put	105.000000	14d	5.39	-0.8151	0.051680	5.2266	-15.3575	26.37%
42	Call	110.000000	14d	0.08	0.0384	0.015887	1.6309	-5.8027	26.76%
43	Put	110.000000	14d	9.95	-0.9616	0.015887	1.6309	-2.5065	26.76%
44	Call	115.000000	14d	0.01	0.0049	0.002674	0.2782	-0.9980	27.12%
45	Put	115.000000	14d	14.88	-0.9951	0.002674	0.2782	2.4480	27.12%
46	Call	120.000000	14d	0.00	0.0004	0.000278	0.0292	-0.1058	27.45%
47	Put	120.000000	14d	19.86	-0.9996	0.000278	0.0292	3.4900	27.45%
48	Call	125.000000	14d	0.00	0.0000	0.000019	0.0021	-0.0076	27.74%
49	Put	125.000000	14d	24.86	-1.0000	0.000019	0.0021	3.7381	27.74%
50	Call	130.000000	14d	0.00	0.0000	0.000001	0.0001	-0.0004	28.01%
51	Put	130.000000	14d	29.85	-1.0000	0.000001	0.0001	3.8951	28.01%
52	Call	70.000000	30d	30.17	1.0000	0.000000	0.0000	-2.0948	23.06%
53	Put	70.000000	30d	0.00	-0.0000	0.000000	0.0000	-0.0000	23.06%
54	Call	75.000000	30d	25.18	1.0000	0.000006	0.0012	-2.2462	23.86%
55	Put	75.000000	30d	0.00	-0.0000	0.000006	0.0012	-0.0018	23.86%
56	Call	80.000000	30d	20.20	0.9994	0.000300	0.0606	-2.4828	24.58%
57	Put	80.000000	30d	0.00	-0.0006	0.000300	0.0606	-0.0888	24.58%
58	Call	85.000000	30d	15.23	0.9898	0.003750	0.7769	-3.7037	25.21%
59	Put	85.000000	30d	0.03	-0.0102	0.003750	0.7769	-1.1600	25.21%
60	Call	90.000000	30d	10.45	0.9327	0.017624	3.7326	-8.3359	25.77%
61	Put	90.000000	30d	0.22	-0.0673	0.017624	3.7326	-5.6425	25.77%
62	Call	95.000000	30d	6.26	0.7738	0.039941	8.6238	-15.9149	26.27%
63	Put	95.000000	30d	1.02	-0.2262	0.039941	8.6238	-13.0719	26.27%
64	Call	100.000000	30d	3.18	0.5281	0.051954	11.4089	-20.0322	26.72%
65	Put	100.000000	30d	2.93	-0.4719	0.051954	11.4089	-17.0395	26.72%
66	Call	105.000000	30d	1.35	0.2888	0.043940	9.7942	-16.9840	27.12%
67	Put	105.000000	30d	6.09	-0.7112	0.043940	9.7942	-13.8418	27.12%
68	Call	110.000000	30d	0.48	0.1273	0.026468	5.9784	-10.3618	27.48%
69	Put	110.000000	30d	10.21	-0.8727	0.026468	5.9784	-7.0700	27.48%
70	Call	115.000000	30d	0.15	0.0462	0.012153	2.7775	-4.8325	27.81%
71	Put	115.000000	30d	14.86	-0.9538	0.012153	2.7775	-1.3910	27.81%
72	Call	120.000000	30d	0.04	0.0142	0.004478	1.0341	-1.8089	28.10%
73	Put	120.000000	30d	19.74	-0.9858	0.004478	1.0341	1.7822	28.10%
74	Call	125.000000	30d	0.01	0.0038	0.001377	0.3209	-0.5647	28.36%
75	Put	125.000000	30d	24.70	-0.9962	0.001377	0.3209	3.1761	28.36%
76	Call	130.000000	30d	0.00	0.0009	0.000364	0.0856	-0.1515	28.60%
77	Put	130.000000	30d	29.68	-0.9991	0.000364	0.0856	3.7389	28.60%
78	Call	70.000000	60d	30.34	0.9999	0.000027	0.0104	-2.0969	23.56%

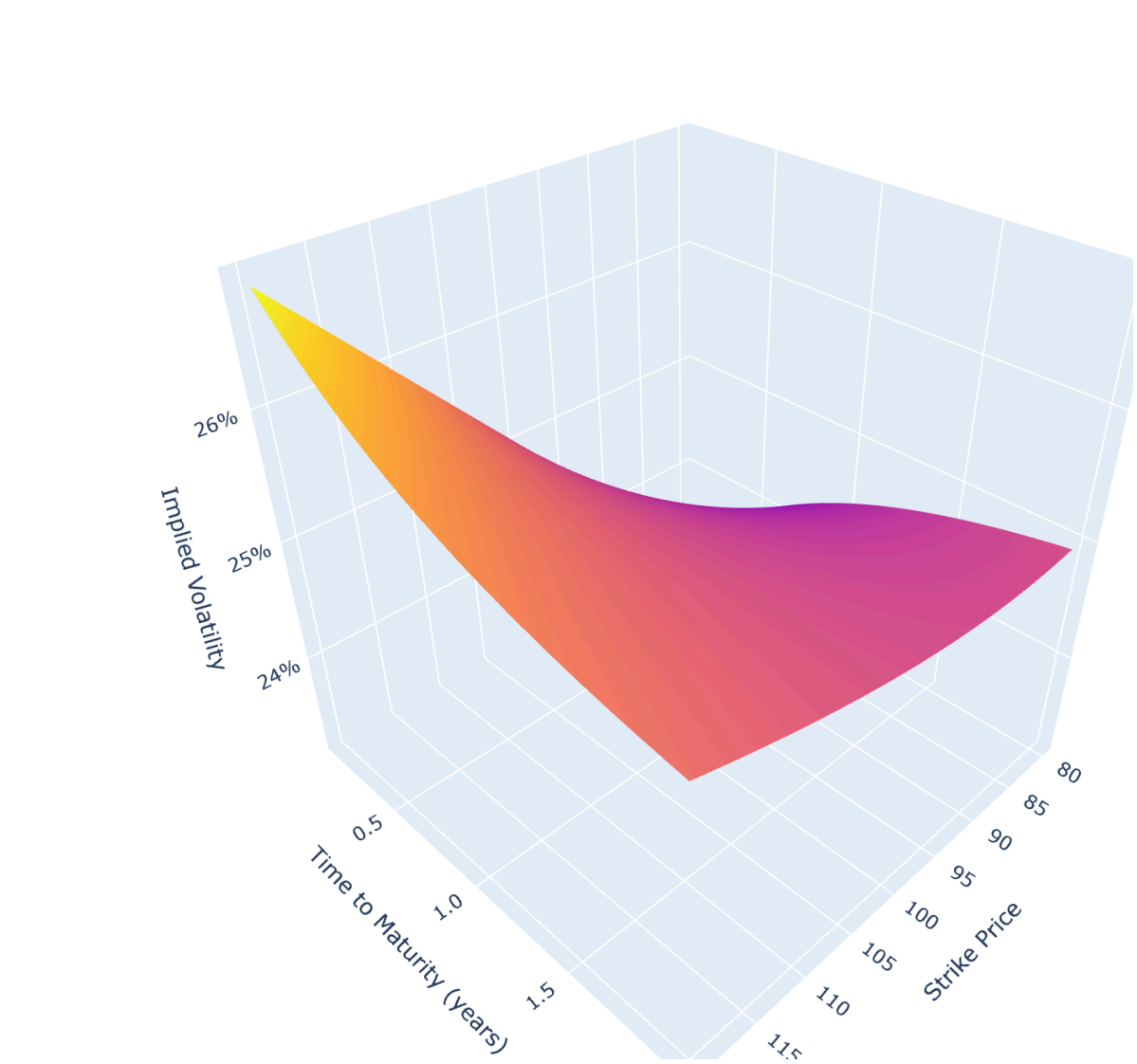
	Type	Strike	Expiry	Price	Delta	Gamma	Vega	Theta	IV
79	Put	70.000000	60d	0.00	-0.0001	0.000027	0.0104	-0.0073	23.56%
80	Call	75.000000	60d	25.37	0.9987	0.000421	0.1681	-2.3591	24.28%
81	Put	75.000000	60d	0.00	-0.0013	0.000421	0.1681	-0.1202	24.28%
82	Call	80.000000	60d	20.43	0.9895	0.002746	1.1244	-3.2074	24.90%
83	Put	80.000000	60d	0.04	-0.0105	0.002746	1.1244	-0.8192	24.90%
84	Call	85.000000	60d	15.63	0.9529	0.009522	3.9847	-5.4756	25.46%
85	Put	85.000000	60d	0.21	-0.0471	0.009522	3.9847	-2.9381	25.46%
86	Call	90.000000	60d	11.20	0.8646	0.020685	8.8226	-9.2205	25.95%
87	Put	90.000000	60d	0.76	-0.1354	0.020685	8.8226	-6.5338	25.95%
88	Call	95.000000	60d	7.44	0.7188	0.031543	13.6770	-12.9064	26.38%
89	Put	95.000000	60d	1.97	-0.2812	0.031543	13.6770	-10.0705	26.38%
90	Call	100.000000	60d	4.57	0.5397	0.036588	16.0946	-14.5824	26.76%
91	Put	100.000000	60d	4.07	-0.4603	0.036588	16.0946	-11.5971	26.76%
92	Call	105.000000	60d	2.59	0.3653	0.034221	15.2443	-13.5837	27.10%
93	Put	105.000000	60d	7.07	-0.6347	0.034221	15.2443	-10.4492	27.10%
94	Call	110.000000	60d	1.36	0.2242	0.026944	12.1361	-10.7465	27.40%
95	Put	110.000000	60d	10.82	-0.7758	0.026944	12.1361	-7.4627	27.40%
96	Call	115.000000	60d	0.67	0.1259	0.018446	8.3896	-7.4182	27.67%
97	Put	115.000000	60d	15.10	-0.8741	0.018446	8.3896	-3.9852	27.67%
98	Call	120.000000	60d	0.31	0.0654	0.011255	5.1629	-4.5691	27.91%
99	Put	120.000000	60d	19.72	-0.9346	0.011255	5.1629	-0.9869	27.91%
100	Call	125.000000	60d	0.14	0.0316	0.006238	2.8833	-2.5568	28.12%
101	Put	125.000000	60d	24.52	-0.9684	0.006238	2.8833	1.1748	28.12%
102	Call	130.000000	60d	0.06	0.0144	0.003188	1.4835	-1.3187	28.30%
103	Put	130.000000	60d	29.42	-0.9856	0.003188	1.4835	2.5621	28.30%
104	Call	70.000000	90d	30.52	0.9996	0.000133	0.0731	-2.1163	22.27%
105	Put	70.000000	90d	0.00	-0.0004	0.000133	0.0731	-0.0318	22.27%
106	Call	75.000000	90d	25.57	0.9960	0.001047	0.5914	-2.4957	22.92%
107	Put	75.000000	90d	0.01	-0.0040	0.001047	0.5914	-0.2623	22.92%
108	Call	80.000000	90d	20.68	0.9791	0.004307	2.4928	-3.5035	23.47%
109	Put	80.000000	90d	0.09	-0.0209	0.004307	2.4928	-1.1212	23.47%
110	Call	85.000000	90d	16.00	0.9316	0.011090	6.5521	-5.4982	23.96%
111	Put	85.000000	90d	0.38	-0.0684	0.011090	6.5521	-2.9670	23.96%
112	Call	90.000000	90d	11.75	0.8393	0.020148	12.1153	-8.1568	24.39%
113	Put	90.000000	90d	1.08	-0.1607	0.020148	12.1153	-5.4767	24.39%
114	Call	95.000000	90d	8.14	0.7050	0.028064	17.1331	-10.4727	24.76%
115	Put	95.000000	90d	2.44	-0.2950	0.028064	17.1331	-7.6437	24.76%
116	Call	100.000000	90d	5.33	0.5484	0.031790	19.6639	-11.4883	25.09%
117	Put	100.000000	90d	4.59	-0.4516	0.031790	19.6639	-8.5104	25.09%
118	Call	105.000000	90d	3.29	0.3953	0.030568	19.1237	-10.9261	25.37%

	Type	Strike	Expiry	Price	Delta	Gamma	Vega	Theta	IV
119	Put	105.000000	90d	7.52	-0.6047	0.030568	19.1237	-7.7994	25.37%
120	Call	110.000000	90d	1.93	0.2652	0.025754	16.2714	-9.1920	25.62%
121	Put	110.000000	90d	11.12	-0.7348	0.025754	16.2714	-5.9163	25.62%
122	Call	115.000000	90d	1.07	0.1667	0.019472	12.4076	-6.9696	25.84%
123	Put	115.000000	90d	15.23	-0.8333	0.019472	12.4076	-3.5450	25.84%
124	Call	120.000000	90d	0.57	0.0988	0.013455	8.6370	-4.8386	26.03%
125	Put	120.000000	90d	19.69	-0.9012	0.013455	8.6370	-1.2651	26.03%
126	Call	125.000000	90d	0.29	0.0555	0.008618	5.5674	-3.1157	26.20%
127	Put	125.000000	90d	24.37	-0.9445	0.008618	5.5674	0.6067	26.20%
128	Call	130.000000	90d	0.14	0.0298	0.005173	3.3606	-1.8803	26.34%
129	Put	130.000000	90d	29.19	-0.9702	0.005173	3.3606	1.9909	26.34%
130	Call	70.000000	180d	31.07	0.9917	0.001415	1.5952	-2.4125	22.86%
131	Put	70.000000	180d	0.05	-0.0083	0.001415	1.5952	-0.3434	22.86%
132	Call	75.000000	180d	26.28	0.9730	0.003799	4.3733	-3.1657	23.34%
133	Put	75.000000	180d	0.18	-0.0270	0.003799	4.3733	-0.9487	23.34%
134	Call	80.000000	180d	21.69	0.9345	0.007648	8.9568	-4.3095	23.75%
135	Put	80.000000	180d	0.51	-0.0655	0.007648	8.9568	-1.9447	23.75%
136	Call	85.000000	180d	17.42	0.8713	0.012416	14.7499	-5.6939	24.09%
137	Put	85.000000	180d	1.17	-0.1287	0.012416	14.7499	-3.1814	24.09%
138	Call	90.000000	180d	13.61	0.7845	0.017091	20.5476	-7.0241	24.38%
139	Put	90.000000	180d	2.29	-0.2155	0.017091	20.5476	-4.3637	24.38%
140	Call	95.000000	180d	10.35	0.6803	0.020672	25.1019	-7.9975	24.62%
141	Put	95.000000	180d	3.95	-0.3197	0.020672	25.1019	-5.1893	24.62%
142	Call	100.000000	180d	7.65	0.5683	0.022545	27.6041	-8.4240	24.83%
143	Put	100.000000	180d	6.19	-0.4317	0.022545	27.6041	-5.4680	24.83%
144	Call	105.000000	180d	5.52	0.4578	0.022598	27.8590	-8.2689	25.00%
145	Put	105.000000	180d	8.98	-0.5422	0.022598	27.8590	-5.1652	25.00%
146	Call	110.000000	180d	3.88	0.3565	0.021121	26.1830	-7.6263	25.14%
147	Put	110.000000	180d	12.27	-0.6435	0.021121	26.1830	-4.3748	25.14%
148	Call	115.000000	180d	2.67	0.2689	0.018609	23.1732	-6.6596	25.25%
149	Put	115.000000	180d	15.98	-0.7311	0.018609	23.1732	-3.2603	25.25%
150	Call	120.000000	180d	1.80	0.1970	0.015589	19.4819	-5.5428	25.34%
151	Put	120.000000	180d	20.03	-0.8030	0.015589	19.4819	-1.9956	25.34%
152	Call	125.000000	180d	1.18	0.1404	0.012500	15.6641	-4.4216	25.41%
153	Put	125.000000	180d	24.35	-0.8596	0.012500	15.6641	-0.7266	25.41%
154	Call	130.000000	180d	0.77	0.0976	0.009644	12.1104	-3.3964	25.46%
155	Put	130.000000	180d	28.86	-0.9024	0.009644	12.1104	0.4464	25.46%
156	Call	70.000000	270d	31.75	0.9746	0.002896	5.1002	-2.7920	23.81%
157	Put	70.000000	270d	0.21	-0.0254	0.002896	5.1002	-0.7381	23.81%
158	Call	75.000000	270d	27.18	0.9445	0.005387	9.6392	-3.5940	24.19%

	Type	Strike	Expiry	Price	Delta	Gamma	Vega	Theta	IV
159	Put	75.000000	270d	0.53	-0.0555	0.005387	9.6392	-1.3934	24.19%
160	Call	80.000000	270d	22.88	0.8979	0.008456	15.3254	-4.5453	24.50%
161	Put	80.000000	270d	1.12	-0.1021	0.008456	15.3254	-2.1980	24.50%
162	Call	85.000000	270d	18.93	0.8350	0.011660	21.3523	-5.5099	24.76%
163	Put	85.000000	270d	2.06	-0.1650	0.011660	21.3523	-3.0159	24.76%
164	Call	90.000000	270d	15.39	0.7585	0.014529	26.8286	-6.3403	24.96%
165	Put	90.000000	270d	3.42	-0.2415	0.014529	26.8286	-3.6995	24.96%
166	Call	95.000000	270d	12.32	0.6730	0.016697	31.0346	-6.9204	25.13%
167	Put	95.000000	270d	5.23	-0.3270	0.016697	31.0346	-4.1330	25.13%
168	Call	100.000000	270d	9.70	0.5835	0.017961	33.5582	-7.1886	25.26%
169	Put	100.000000	270d	7.50	-0.4165	0.017961	33.5582	-4.2545	25.26%
170	Call	105.000000	270d	7.52	0.4948	0.018291	34.3091	-7.1393	25.36%
171	Put	105.000000	270d	10.21	-0.5052	0.018291	34.3091	-4.0585	25.36%
172	Call	110.000000	270d	5.74	0.4110	0.017785	33.4546	-6.8110	25.43%
173	Put	110.000000	270d	13.33	-0.5890	0.017785	33.4546	-3.5834	25.43%
174	Call	115.000000	270d	4.33	0.3347	0.016620	31.3228	-6.2685	25.48%
175	Put	115.000000	270d	16.80	-0.6653	0.016620	31.3228	-2.8942	25.48%
176	Call	120.000000	270d	3.22	0.2675	0.015003	28.3079	-5.5866	25.51%
177	Put	120.000000	270d	20.58	-0.7325	0.015003	28.3079	-2.0656	25.51%
178	Call	125.000000	270d	2.36	0.2101	0.013137	24.7974	-4.8366	25.52%
179	Put	125.000000	270d	24.62	-0.7899	0.013137	24.7974	-1.1689	25.52%
180	Call	130.000000	270d	1.71	0.1623	0.011195	21.1253	-4.0785	25.51%
181	Put	130.000000	270d	28.86	-0.8377	0.011195	21.1253	-0.2641	25.51%
182	Call	70.000000	365d	32.50	0.9589	0.003686	8.8015	-2.9527	23.88%
183	Put	70.000000	365d	0.43	-0.0411	0.003686	8.8015	-0.9147	23.88%
184	Call	75.000000	365d	28.13	0.9242	0.005898	14.2707	-3.6553	24.20%
185	Put	75.000000	365d	0.91	-0.0758	0.005898	14.2707	-1.4718	24.20%
186	Call	80.000000	365d	24.03	0.8765	0.008349	20.4129	-4.4039	24.45%
187	Put	80.000000	365d	1.67	-0.1235	0.008349	20.4129	-2.0748	24.45%
188	Call	85.000000	365d	20.28	0.8171	0.010753	26.5052	-5.1093	24.65%
189	Put	85.000000	365d	2.77	-0.1829	0.010753	26.5052	-2.6347	24.65%
190	Call	90.000000	365d	16.91	0.7485	0.012853	31.8785	-5.6915	24.80%
191	Put	90.000000	365d	4.25	-0.2515	0.012853	31.8785	-3.0713	24.80%
192	Call	95.000000	365d	13.93	0.6739	0.014463	36.0391	-6.0939	24.92%
193	Put	95.000000	365d	6.12	-0.3261	0.014463	36.0391	-3.3281	24.92%
194	Call	100.000000	365d	11.35	0.5968	0.015486	38.7147	-6.2892	25.00%
195	Put	100.000000	365d	8.39	-0.4032	0.015486	38.7147	-3.3779	25.00%
196	Call	105.000000	365d	9.14	0.5200	0.015903	39.8438	-6.2771	25.05%
197	Put	105.000000	365d	11.04	-0.4800	0.015903	39.8438	-3.2202	25.05%
198	Call	110.000000	365d	7.29	0.4463	0.015760	39.5326	-6.0784	25.08%

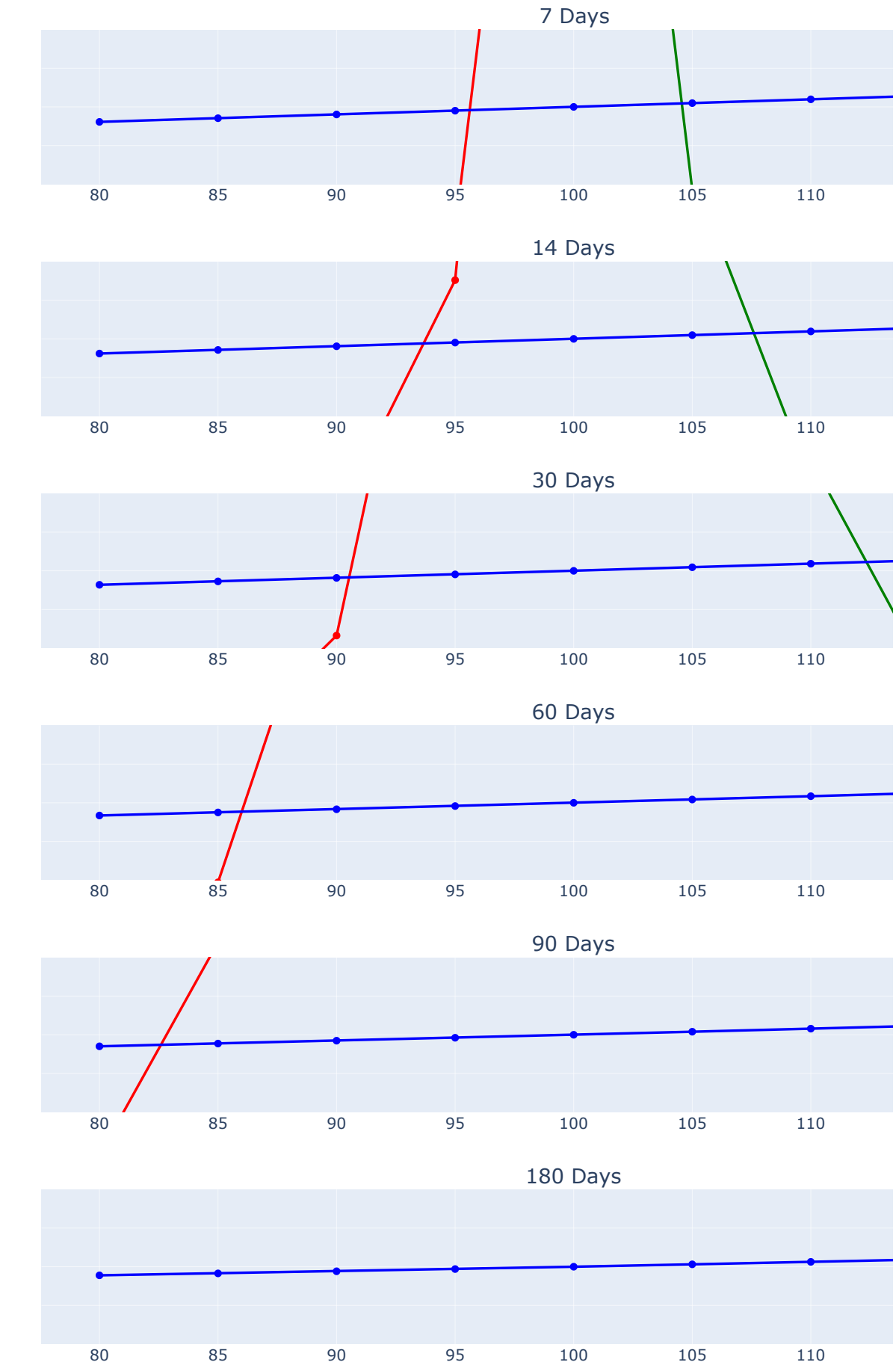
	Type	Strike	Expiry	Price	Delta	Gamma	Vega	Theta	IV
199	Put	110.000000	365d	14.04	-0.5537	0.015760	39.5326	-2.8759	25.08%
200	Call	115.000000	365d	5.75	0.3775	0.015144	37.9991	-5.7273	25.09%
201	Put	115.000000	365d	17.35	-0.6225	0.015144	37.9991	-2.3793	25.09%
202	Call	120.000000	365d	4.49	0.3149	0.014163	35.5203	-5.2644	25.08%
203	Put	120.000000	365d	20.95	-0.6851	0.014163	35.5203	-1.7708	25.08%
204	Call	125.000000	365d	3.47	0.2592	0.012928	32.3879	-4.7306	25.05%
205	Put	125.000000	365d	24.78	-0.7408	0.012928	32.3879	-1.0914	25.05%
206	Call	130.000000	365d	2.66	0.2107	0.011546	28.8764	-4.1634	25.01%
207	Put	130.000000	365d	28.82	-0.7893	0.011546	28.8764	-0.3786	25.01%

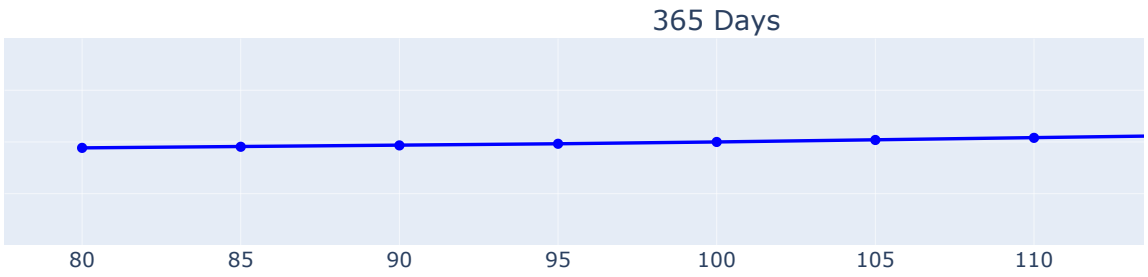
In [14]:
```
def plot_volatility_surface_interactive():
    strikes = np.linspace(80, 120, 25)
    maturities = np.linspace(0.1, 2, 25)
    X, Y = np.meshgrid(strikes, maturities)

    Z = 0.25 + 0.1*(np.log(X/100)) * np.exp(-Y) + 0.05*(np.log(X/100))**2

    fig = go.Figure(data=[go.Surface(z=Z, x=X, y=Y)])

    fig.update_layout(
        title='3D Implied Volatility Surface',
        scene=dict(
            xaxis_title='Strike Price',
            yaxis_title='Time to Maturity (years)',
            zaxis_title='Implied Volatility',
            xaxis=dict(tickformat=".0f"),
            yaxis=dict(tickformat=".1f"),
            zaxis=dict(tickformat=".0%")
        ),
        width=1000,
        height=800,
        margin=dict(l=65, r=50, b=65, t=90)
    )

    return fig

plot_volatility_surface_interactive().show()
```



In [16]:
```python
def display_compact_options_chain(S0=100):
    strikes = np.arange(S0*0.8, S0*1.2+1, 5)
    maturities = [7, 14, 30, 60, 90, 180, 365]

    fig = make_subplots(
        rows=len(maturities), cols=1,
        subplot_titles=[f"{d} Days" for d in maturities],
        vertical_spacing=0.05
    )

    for i, days in enumerate(maturities):
        T = days/365
        calls = []
        puts = []
        ivs = []

        for K in strikes:
            vol = 0.25 + 0.1*(np.log(K/S0)) * np.exp(-T) + 0.05*(np.log(K/S0))**2
            call_price, _ = black_scholes(S0, K, T, 0.03, vol, 'call')
```

```
            put_price, _ = black_scholes(S0, K, T, 0.03, vol, 'put')

            calls.append(call_price)
            puts.append(put_price)
            ivs.append(vol)

        fig.add_trace(
            go.Scatter(
                x=strikes, y=calls,
                name='Calls', line=dict(color='green')),
            row=i+1, col=1
        )
        fig.add_trace(
            go.Scatter(
                x=strikes, y=puts,
                name='Puts', line=dict(color='red')),
            row=i+1, col=1
        )
        fig.add_trace(
            go.Scatter(
                x=strikes, y=ivs,
                name='IV', line=dict(color='blue'), yaxis='y2'),
            row=i+1, col=1
        )

        fig.update_yaxes(title_text="Option Price", row=i+1, col=1)
        fig.update_yaxes(
            title_text="Implied Vol",
            range=[0.15, 0.35],
            overlaying='y',
            side='right',
            row=i+1, col=1
        )

    fig.update_layout(
        height=200*len(maturities),
        width=1000,
        title_text=f"Options Chain Analysis (Spot = {S0})",
        showlegend=False
    )

    return fig

display_compact_options_chain().show()
```


Options Chain Analysis (Spot = 100)
7 Days
14 Days
30 Days
60 Days
90 Days
180 Days
80
85
90
95
100
105
110



Key Observations

1. **Volatility Surface Patterns**
   - Clear smile/skew across maturities (steepest for short-dated options)
   - ATM gamma peaks near expiry → Higher delta hedging frequency needed
2. **Time Decay Dynamics**
   - Theta decay most severe for ATM options (accelerated in final 30 days)

Trader Playbook

Situation	Action	Rationale
Trading long-dated options	Vega hedging + Monthly rebalance	Manages volatility drift over time
High gamma environment	Daily delta rebalancing	Controls nonlinear price risks
Volatility surface shifts	Adjust strike selection	Exploits skew/smile mispricing

Conclusion

Optimal Hedging Strategies Under Stochastic Volatility

Strategy Performance Summary

Market Condition	Optimal Strategy	Performance Edge	Key Risks
Stable Volatility	Delta Hedging	Lowest tracking error (RMSE: 1.94)	Fails during volatility jumps
Trending Volatility	Delta-Vega Hybrid	67% lower mean error vs delta-only	Over-hedging in mean-reversion
Volatility Spikes	Regime-Switching Alerts	Early warning for adjustments	1-2 day detection lag
Long-Dated Options	Monthly Vega Rebalancing	Captures term structure shifts	Gamma exposure near expiry

Critical Limitations

- **Model Boundaries**: Continuous hedging assumption, no transaction costs
- **Extreme Markets**: Untested when VIX > 50 or correlations break
- **Implementation**: Regime detection requires real-time monitoring

Final Recommendations

> "Prioritize delta-vega hybrids for >60-day options, but taper vega exposure when IV percentile < 20%."

In [ ]: